UNIVERSITAT JAUME·I

DESARROLLO DE UNA INTERFAZ WEB PARA LA COMPOSICIÓN DE SERVICIOS REST

Máster en Sistemas Inteligentes
Universitat Jaume I
Curso 2014/2015

SIU043 – Trabajo fin de máster

Autora: Sara Valero Menacho
Tutor: Ismael Sanz Blasco

Resumen

El creciente aumento del uso de servicios web REST se debe a la simplicidad y ligereza que ofrece este estilo arquitectónico. Este incremento propicia la necesidad de reutilizar los servicios existentes para componer servicios nuevos. Sin embargo, el proceso de composición es un proceso complejo que resulta difícil de automatizar por lo que mayoría de enfoques proponen métodos manuales o semiautomáticos.

En este documento se presenta la memoria del trabajo de fin de máster perteneciente al máster de sistemas inteligentes de la Universitat Jaume I. El proyecto consiste en el desarrollo de una interfaz web que a partir de un repositorio de servicios web REST permite gestionar sus descripciones así como componer manualmente un nuevo servicio reutilizando algunos de los existentes.

El presente documento describe el proceso que se ha seguido a la hora de realizar el proyecto. En primer lugar, se establece el contexto y la motivación, se indican los objetivos, y se muestra la planificación temporal, la tecnología usada y la arquitectura del sistema.

Seguidamente se hace un pequeño estado del arte sobre la composición de servicios web en el que se muestra una taxonomía de los métodos y técnicas que existen para componer. Además, también se habla sobre REST y los servicios web.

A continuación se definen los requisitos que tiene que cumplir la aplicación, se detalla el diseño y la implementación realizados así como las pruebas. También se expone un escenario de uso para ver el correcto funcionamiento de la interfaz.

Finalmente se presentan las conclusiones y el trabajo futuro.

Palabras clave

Composición de servicios, descripción de un servicio, servicio web, REST.

Agradecimientos

A mi padre, cuyo recuerdo siempre me acompaña.

A mi madre, por su amor desinteresado.

A mi hermano, por su cariño puro e inocente.

A Manu, por su apoyo incondicional.

A mi tutor Ismael, por su paciencia y buenos consejos.

Índice general

Índice de figuras

Índice de tablas

1 INTRODUCCIÓN

Este capítulo describe en primer lugar el contexto en el que se situa este trabajo, así como su motivación y los objetivos que se quieren alcanzar. A continuación, se presenta la planificación temporal del proyecto y se explican las tecnologías utilizadas en el desarrollo de la interfaz y la arquitectura del sistema. El capítulo concluye con la organización del resto del trabajo.

1.1 Contexto

Según el W3C la *WWW (World Wide Web)* es un espacio de información en el que los elementos de interés, conocidos como recursos, se identifican mediante identificadores globales denominados *Uniform Resource Identifier (URI)*. Además de URI, las principales tecnologías que conforman la red son *HTTP (Hypertext Transfer Protocol)* y *HTML (HyperText Markup Language)*. La Web fue creada en 1989 por Tim Berners-Lee y Robert Cailliau, aunque no se abrió hasta el 30 de abril de 1993, cuando el CERN anunció de forma oficial mediante la publicación de un documento que la web sería de dominio público, abierta para todos y sin costes.

A medida que crecía el uso de la WWW fueron apareciendo distintas plataformas (PC, Mainframe, Mac, etc.) y lenguajes de programación (PHP, C#, Java, etc.) que necesitaban comunicarse entre sí. Para que plataformas diferentes y con distintos lenguajes pudieran comunicarse era necesario estandarizar la comunicación entre ellas, naciendo así los servicios web.

Los servicios web proporcionan mecanismos de comunicación estándar entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Si bien los servicios web facilitan la comunicación entre distintas plataformas el problema surge cuando intentamos ofrecer servicios (aplicaciones) a través de la Web. Mientras que la Web se caracteriza por su simplicidad, los servicios web que se han venido utilizando tienen debilidades tales como una arquitectura muy pesada para el acceso a objetos distribuidos remotos, complejidad elevada y un proceso de estandarización problemático.

Debido a estos problemas se pensó que, ya que las características de la web han hecho que ésta sea lo que es actualmente, se podría utilizar la web tal y como fue concebida. Es decir, las mismas características que hacen que una web sea fácil de usar por un navegador también hacen que la API de un servicio web sea fácil de usar por un programador. La idea es ajustar todos los servicios web de forma que respeten la arquitectura de la Web y se aprovechen de sus ventajas. Para poder aprovechar las características de la Web se diseñó *REST (Representational State Transfer)* que es un estilo arquitectónico especialmente diseñado para los sistemas distribuidos de hipermedios cuyo término acuñó Roy Thomas Fielding en su tesis doctoral [5]. Actualmente, el número de servicios web basados en REST está creciendo y cada vez más empresas se decantan por este estilo arquitectónico debido a su mayor ligereza y simplicidad.

1.2 Motivación

Tal y como se ha visto en el apartado anterior, el uso de servicios web basados en REST está aumentando y por tanto se hace cada vez más necesario el uso de mecanismos que permitan su composición. La composición de servicios web permite sintetizar un nuevo servicio mediante la reutilización de servicios existentes. Sin embargo, este proceso de composición resulta complejo de automatizar, por lo que la mayoría de enfoques proponen soluciones manuales o semiautomáticas. Además, en muchos casos este proceso se realiza con servicios web basados en arquitecturas más complejas y pesadas como SOAP.

Teniendo en cuenta este escenario, el uso de servicios web basados en REST se presenta como una solución ideal para aprovechar las características que han hecho que la Web tenga tanto éxito y además permitir su composición.

Aunque existen múltiples enfoques para la composición de servicios web, no ocurre lo mismo cuando se trata de servicios web basados en el estilo arquitectónico REST. Por tanto, la motivación de este trabajo de fin de máster surge de la necesidad de diseñar e implementar una interfaz web que permita realizar la composición manual de servicios web REST y que también permita la gestión de las descripciones de estos servicios.

1.3 Objetivos

El objetivo principal de este trabajo de fin de máster es diseñar e implementar una interfaz web que deberá permitir:

- Gestionar las descripciones de los servicios web REST que se quieran utilizar en la interfaz.
- La composición manual de servicios web basados en el estilo arquitectónico REST.
- La ejecución de los servicios que forman el nuevo servicio compuesto.

1.4 Planificación

En este apartado se presenta la planificación temporal de las tareas que se han de realizar, de acuerdo a los objetivos marcados, para el desarrollo de este trabajo de fin de máster. En la tabla que se muestra a continuación se identifican y explican cada una de estas actividades.

Tarea	Horas planificadas	Objetivo
Documentación sobre tecnologías web	50	Aprender a utilizar las tecnologías web necesarias para desarrollar la interfaz
Análisis del proceso de composición de servicios	20	Comprender los requisitos que debe cumplir la interfaz para el proceso de composición de servicios
Análisis de la gestión de descripciones de servicios que se realizará mediante la interfaz	20	Comprender que requisitos debe cumplir la interfaz para permitir la gestión de las descripciones de los servicios
Diseño de la interfaz para la composición de servicios	20	Decidir cómo se va a desarrollar el proceso de composición y qué pasos se van a seguir
Diseño de la interfaz para la gestión de descripciones de los servicios	20	Decidir cómo va a funcionar la interfaz para realizar la gestión de descripciones de los servicios usando la API del servidor
Implementación de la interfaz para el proceso de composición	50	Desarrollar la parte de la interfaz que permite realizar la composición de los servicios
Implementación de la interfaz para la gestión de las descripciones de los servicios	50	Desarrollar la parte de la interfaz que permite realizar la gestión de las descripciones de los servicios usando la API del servidor
Pruebas de la interfaz	20	Depurar errores del proceso de composición y de la gestión de descripciones de servicios
Redacción de la memoria	40	Escribir la memoria del TFM para su evaluación
Elaboración de la presentación	10	Preparar la presentación del TFM para exponerla

Tabla 1: Planificación temporal

Como se puede ver en la tabla, el tiempo total son 300 horas tal y como se estipula en la normativa de la asignatura SIU043 Trabajo de fin de máster. Sin embargo ha habido algunas desviaciones respecto a esta planificación inicial. Por una parte, se ha tenido que dedicar más tiempo del previsto a aprender a utilizar algunas de las tecnologías utilizadas, en concreto JavaScript y la librería para uso de grafos Cytoscape.js. Y por otra parte, durante la fase de implementación se han detectado requisitos que no habían sido definidos en la fase de análisis y se ha tenido que volver atrás en el ciclo de desarrollo. Además, durante la fase de pruebas se han detectado fallos que han supuesto una mayor dedicación para solucionarlos.

1.5 Tecnologías empleadas

Para el desarrollo de la interfaz se han utilizado una serie de tecnologías que abarcan desde el diseño de la interfaz hasta su funcionalidad. A continuación se detallan estas tecnologías y se describen brevemente.

- **HTML5:** lenguaje de marcado que permite estructurar y organizar el contenido de la interfaz web. Además en su última versión permite realizar nuevas funcionalidades como por ejemplo la API para hacer *Drag* & *Drop* (arrastrar y soltar) mediante eventos, que se ha utilizado en el desarrollo de esta interfaz.

- **CSS3:** lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML que permite aplicar estilos de forma rápida.

- **BOOTSTRAP**: framework para diseño de sitios y aplicaciones web. Contiene plantillas de diseño basado en HTML y CSS con diferentes elementos como formularios, botones, etc.

- **JavaScript:** lenguaje del lado del cliente que permite añadir funcionalidad a la interfaz desarrollada. La principal librería JavaScript que se ha utilizado en este proyecto es Cytoscape.js para el desarrollo de la parte gráfica de la composición de servicios. Además también se han utilizado otras librerías JavaScript como jQuery y jQuery UI y AJAX para el acceso a los datos mediante XMLHttpRequest.

    - **Cytoscape.js**: librería JavaScript para el análisis y visualización de grafos que se ha utilizado para la creación de los grafos de composición de servicios.
    - **jQuery:** librería JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.
    - **jQuery UI:** librería JavaScript de componentes para el framework jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales para la creación de aplicaciones web. En concreto se ha utilizado para el desarrollo de las ventanas de diálogo de la interfaz de composición.
    - **AJAX:** técnica de desarrollo web para crear aplicaciones interactivas que se ejecutan en el navegador del usuario mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. Normalmente se utiliza con JavaScript para efectuar funciones de llamada mientras que el acceso a los datos se realiza mediante XMLHttpRequest. Estas llamadas AJAX con XMLHttpRequest se han utilizado para la comunicación con los servicios y con la API REST del servidor.

- **HTTP:** protocolo para la comunicación con el servidor que mediante sus métodos de petición (también conocidos como "verbos") permite indicar la acción que desea que se efectúe sobre el recurso identificado. En el desarrollo de esta interfaz principalmente se han utilizado los verbos GET, PUT y DELETE.

- **JSON:** formato ligero para el intercambio de datos. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. En la interfaz se utiliza JSON para intercambiar información tanto con el servidor como con los servicios.

- **REST:** estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. En la actualidad se usa en el sentido más amplio para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP. Tanto la interfaz desarrollada como la API y los

servicios web que se consideran se basan en esta arquitectura. En el estado del arte se verá con más detalle qué es REST y en qué consiste.

1.6 Arquitectura del sistema

En la figura 1 se muestra la arquitectura general del sistema en el que se va a implementar la interfaz. El sistema está formado por la interfaz web y por el servidor y el repositorio de servicios. La interfaz consta de dos módulos, uno para la gestión de descripciones de servicios web y otro para la composición de servicios. Esta interfaz se comunica con el servidor y con el repositorio de servicios mediante peticiones HTTP (GET, PUT, DELETE). En el caso del servidor la comunicación se realiza utilizando la API REST que éste proporciona. Además, el intercambio de información entre la interfaz, el servidor y el repositorio se realiza usando el formato JSON.

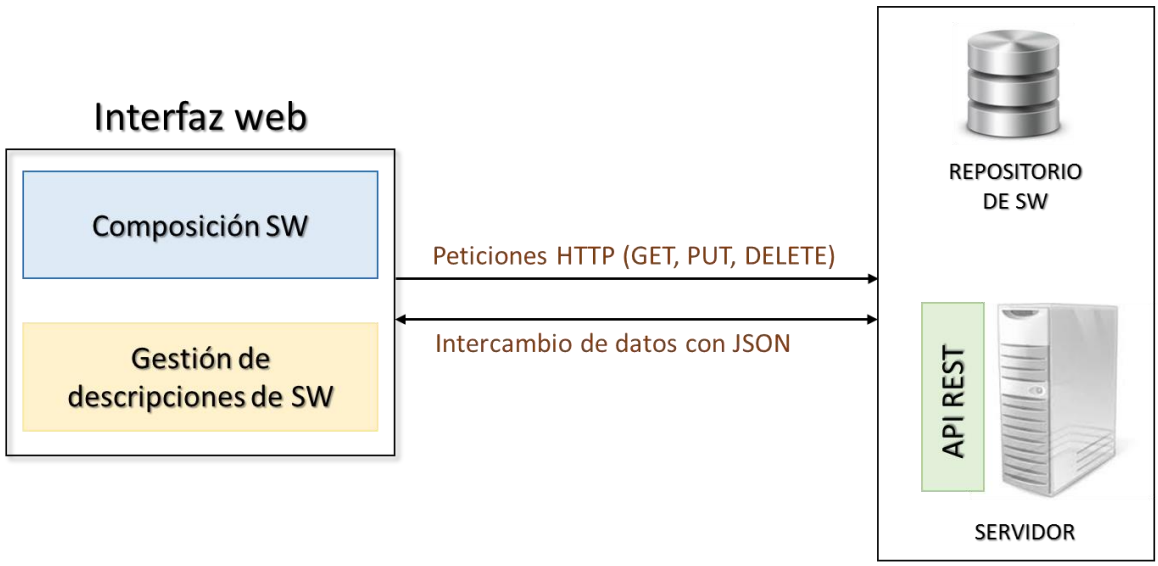


Figura 1: Arquitectura del sistema

1.7 Estructura de la memoria

Los restantes capítulos de este trabajo se organizan de la siguiente manera:

El capítulo 2 presenta el estado del arte de la composición de servicios web. En primer lugar, introduce brevemente qué es el estilo arquitectónico REST y cúales son sus principios de diseño. También define lo que son los servicios web y explica sus principales formas de desarrollo. A continuación, se habla sobre la composición y se presenta el estado del arte de los métodos y técnicas de composición de servicios web que existen resumidos en una taxonomía. También se comentan distintos trabajos relacionados tanto con la composición de servicios web como con la composición de servicios web REST.

El capítulo 3 consiste en el desarrollo de la interfaz. Primero se definen los requisitos que tienen que cumplir tanto el módulo de gestión de descripciones como el módulo de composición. A continuación se explica el diseño y la implementación realizados para cumplir con los objetivos marcados. Finalmente, se presentan las pruebas realizadas y se expone un escenario de uso para demostrar el correcto funcionamiento de la interfaz.

Por último, en el capítulo 4 se resume lo visto en este trabajo, se presentan las conclusiones y el trabajo futuro y en el anexo I se presenta el manual de usuario de la interfaz web desarrollada.

2 ESTADO DEL ARTE

En este capítulo se presenta en primer lugar una breve introducción a la arquitectura REST, en la que se basan los servicios web que se usan en este proyecto. A continuación, se define el concepto de servicio web y se comentan las principales formas de desarrollo de servicios web, los servicios web basados en SOAP y los basados en REST. Finalmente, en el último apartado, se habla sobre el tema de la composición de servicios web, se explica en qué consiste y se presenta una taxonomía de los métodos y técnicas que existen. También se mencionan algunos trabajos relacionados con la composición de servicios web REST.

2.1 REST (Representational State Transfer)

En este apartado se define qué es REST, y se presenta su motivación y objetivos. A continuación se describen los principios de diseño que rigen esta arquitectura.

2.1.1 Definición

REST (Representational State Transfer) es un estilo de arquitectura software para sistemas hipermedia distribuidos como la WWW, que consta de una serie de principios de diseño o restricciones arquitectónicas y que no tiene en cuenta los detalles de implementación. REST es ligero y simple y nos permite aprovechar las características de la Web. Debido a las ventajas que ofrece como una mayor escalabilidad o el uso de interfaces uniformes ha aumentado su uso en detrimento del protocolo SOAP más pesado y complejo.

El término REST fue introducido por primera vez en la tesis doctoral de Roy Fielding, uno de los principales autores de la especificación de HTTP, en el 2000. Fielding usó REST para diseñar HTTP 1.1 y los identificadores de recursos uniformes (URI). El estilo arquitectónico REST también se aplica al desarrollo de servicios web como una alternativa a otras especificaciones de computación distribuida como SOAP. Es importante destacar que REST no es un estándar, aunque está basado en estándares tales como HTTP, URL, XML, etc.

La motivación de REST era aprovechar las ventajas (escalabilidad, generalidad de interfaces, desarrollo independiente de componentes y existencia de proxys) que han hecho que la Web tenga tanto éxito mediante un modelo de arquitectura [3]. Además, se pretendía que REST pudiese servir como marco de trabajo para los estándares de protocolos Web.

Los objetivos [4] que persigue REST son conseguir la escalabilidad de los componentes de interacción y la generalidad de las interfaces. También pretende conseguir la independencia en el desarrollo de componentes y utilizar sistemas intermedios para reducir el tiempo de interacción, mejorar la seguridad, y encapsular los sistemas de herencia.

2.1.2 Principios de diseño

Para poder conseguir los objetivos que se han comentado en el apartado anterior, REST establece una serie de principios de diseño que se deben cumplir [5].

El primer principio es la **identificación única de recursos**. Los recursos se deben identificar y manipular a través de representaciones. Esto se consigue mediante el uso de URIs. HTTP es un protocolo centrado en URIs. Los recursos son los objetos lógicos a los que se le envían mensajes. Los recursos no pueden ser directamente accedidos o modificados. Más bien se trabaja con representaciones de ellos. Internamente el estado del recurso puede ser cualquier cosa, desde una base de datos relacional a un fichero de texto.

El segundo principio es el de **interfaz uniforme**, en el que todos los recursos deben compartir la misma interfaz uniforme formada por un conjunto de operaciones limitadas (ver tabla 2) para transferencia de estado (en HTTP GET, PUT, POST, DELETE) y por un conjunto limitado de tipos de contenidos (en HTTP se identifican mediante tipos MIME: XML, HTML,…)

MÉTODO	FUNCIÓN
GET	Solicitar recurso
POST	Crear recurso nuevo
PUT	Actualizar o modificar recurso
DELETE	Borrar recurso

Tabla 2: Métodos HTTP

El tercer principio es el de **mensajes autodescriptivos**. REST dicta que los mensajes HTTP deberían ser tan descriptivos como sea posible. Esto hace posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario. Uno de los modos de que HTTP logre esto es por medio del uso de varios métodos estándares, muchos encabezamientos y un mecanismo de direccionamiento. HTTP es un protocolo sin estado y, cuando se utiliza adecuadamente, es posible interpretar cada mensaje sin ningún conocimiento de los mensajes precedentes.

El cuarto principio es el de **hipermedia como un mecanismo del estado de la aplicación**, comúnmente conocido como *HATEOAS (Hypermedia As The Engine Of Application State)*. Este principio dice que el estado actual de una aplicación web debería ser capturado en uno o más documentos de hipertexto, residiendo tanto en el cliente como en el servidor. El servidor conoce el estado de sus recursos, aunque no intenta mantener las sesiones individuales de los clientes. Esta es la misión del navegador, el sabe como navegar de recurso a recurso, recogiendo información que el necesita o cambiando el estado que el necesita cambiar. El formato de las representaciones debe estar documentado y ser estándar. Las representaciones deben incluir enlaces a otros recursos relacionados.

El quinto y último principio es el de **comunicación sin estado** (stateless). Según este principio no se debe establecer ninguna sesión entre el cliente y el servidor. El estado del cliente es gestionado por el cliente y el estado de los recursos es gestionado por el servidor. El servidor no guarda el estado de la comunicación.

Si un servicio web se ajusta a los principios de diseño de la arquitectura REST se dice que el servicio es *RESTful*. En el caso de que se viole cualquiera de las restricciones necesarias, ya no puede considerarse RESTful.

2.2 Servicios Web

En este apartado se define lo qué es un servicio web y cómo funcionan. A continuación se comparan los servicios web SOAP con los servicios web REST.

2.2.1 Definición

Según la Wikipedia [6], un servicio web es "una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos."

Según el W3C [7] un servicio web es "un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios.

Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web."

Desde un punto de vista general, un servicio web es un componente de software independiente de la plataforma e implementación, que lleva a cabo un servicio concreto y que puede integrarse con otros servicios web para dar un servicio diferente. Este componente de software podrá ser descrito usando un lenguaje de descripción de servicios, publicado en un registro de servicios, descubierto a través de un mecanismo estándar, invocado a través de un API y compuesto con otros servicios.

Se trata de un recurso residente en la web, con una dirección URL accesible y que devuelve información al cliente que quiera utilizarlo, pero los detalles de implementación y despliegue del servicio web no son relevantes para el programa que invoca el servicio.

La figura 2 muestra cómo interactúa un conjunto de servicios web:

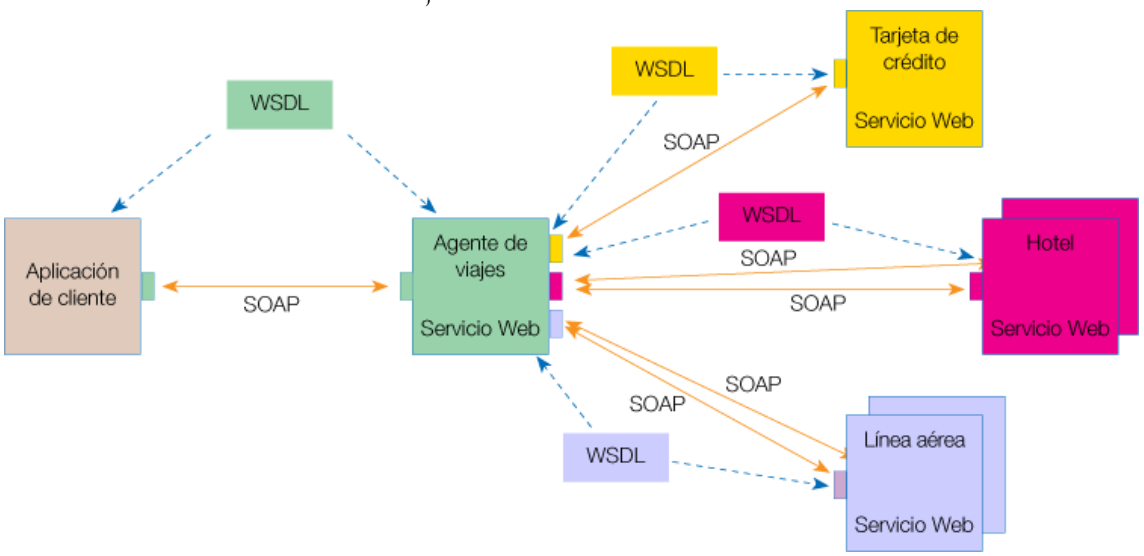


Figura 2: Los servicios web en funcionamiento

Según el ejemplo del gráfico, un usuario (que juega el papel de cliente dentro de los servicios web), a través de una aplicación, solicita información sobre un viaje que desea realizar haciendo una petición a una agencia de viajes que ofrece sus servicios a través de Internet. La agencia de viajes ofrecerá a su cliente (usuario) la información requerida. Para proporcionar al cliente la información que necesita, esta agencia de viajes solicita a su vez información a otros recursos (otros servicios web) en relación con el hotel y la compañía aérea. La agencia de viajes obtendrá información de estos recursos, lo que la convierte a su vez en cliente de esos otros servicios web que le van a proporcionar la información solicitada sobre el hotel y la línea aérea. Por último, el usuario realizará el pago del viaje a través de la agencia de viajes que servirá de intermediario entre el usuario y el servicio web que gestionará el pago.

2.2.2 Servicios Web REST vs Servicios Web SOAP

Hay principalmente dos formas para el desarrollo de servicios web: los servicios web basados en SOAP tradicionales y los servicios web basados en REST, conceptualmente más simples.

Los servicios web REST son unos servicios de software que se publican en la Web, y que aprovechan al máximo las ventajas de usar correctamente el protocolo HTTP. Tal y como su nombre indica utilizan la arquitectura REST que, aunque no es un estándar, sí se basa en estándares [8].

SOAP es un protocolo para el intercambio de mensajes sobre redes de ordenadores, generalmente usando HTTP. Está basado en XML, lo que facilita la lectura, pero también los mensajes resultan más largos y, por lo tanto,

considerablemente más lentos de transferir. Los servicios web basados en SOAP son independientes del protocolo y con estado, pero exigen más recursos de computación, en especial para la manipulación de los mensajes SOAP. Estos servicios web se utilizan normalmente para integrar aplicaciones empresariales complejas.

Por el contrario, los servicios web RESTful utilizan el modelo REST. Los servicios web RESTful se identifican mediante URIs, que ofrecen un espacio de direccionamiento global de los recursos y de descubrimiento de servicios. Interactúan a través de una interfaz uniforme, que comprende un conjunto fijo de operaciones en el contexto de la Web y el protocolo HTTP. Estos servicios interactúan mediante el intercambio de mensajes de solicitud y respuesta, cada uno de los cuales incluye suficiente información para describir la forma de procesar el mensaje. En contraste con los servicios web basados en SOAP, los servicios web RESTful son ligeros y sin estado, y son muy adecuados para la integración táctica ad hoc a través de Internet [9].

En la actualidad se está llegando a la conclusión que SOAP es demasiado complicado y por este motivo se están comenzando a utilizar servicios web basados en REST para mostrar cantidades de datos masivos.

2.3 Composición de Servicios Web

En este apartado se presentan algunos trabajos relacionados con la composición de servicios web y a continuación se define el problema de la composición de acuerdo con [10], donde se usa una técnica basada en grafos. También, se muestra una taxonomía general sobre la composición de servicios web en la que se ilustran los patrones, métodos y técnicas de composición que hay. Después, se habla sobre la composición de servicios web RESTful y se comentan algunos trabajos sobre composición de servicios basados en REST así como algunos sistemas existentes que realizan una función similar a la que se pretende conseguir en el desarrollo de este proyecto. Por último, se presentan las conclusiones.

2.3.1 Trabajo relacionado

Los recientes progresos en el ámbito de los servicios web han hecho que sea posible publicar, localizar e invocar aplicaciones a través de la Web. La capacidad para la selección e integración eficientes de servicios entre organizaciones heterogéneas en la Web en tiempo de ejecución se convierte en un requisito importante para la prestación de los servicios web. En particular, si un servicio web simple no puede satisfacer la funcionalidad requerida por un usuario, debería existir la posibilidad de combinar los servicios existentes en conjunto con el fin de cumplir con la solicitud. Esta tendencia ha dado lugar a un número considerable de esfuerzos de investigación sobre la composición de servicios web tanto en el mundo académico como en la industria.

En esta sección se presentan algunos trabajos relacionados con la composición de servicios web en los que se hacen estudios de los métodos de composición y se proponen diferentes clasificaciones.

En [9], se presenta un estudio comparativo sobre enfoques de composición de servicios web clasificados según si la composición se realiza de forma automática o semiautomática. También se puede ver en este artículo un breve estudio comparativo de estándares de composición de servicios web categorizados de acuerdo al nivel de automatización del modelado de la automatización de servicios.

En [11], se propone que los métodos de composición de servicios web se clasifiquen en tres grupos: automáticos, semiautomáticos y manuales. Presenta un estudio comparativo de distintos enfoques en el que se tienen en cuenta el tipo de composición (sintáctica o semántica), la manera de hacer la composición (automática, semiautomática o manual), la estrategia de la composición (estática o dinámica), el lenguaje de descripción, el proceso de unión (en tiempo de diseño o en tiempo de ejecución), las herramientas y el modelo.

Según [12], las dos principales categorías de métodos de composición de servicios web son (1) métodos heurísticos y (2) métodos de programación matemática. Entre los métodos de programación matemática, los métodos más populares son los de programación entera (integer programming) y los de programación dinámica (dynamic

programming). Dentro de cada subcategoría de estos métodos de composición, hay clases más pequeñas de métodos para planificación de composición de servicios.

Tal y como propone [13], las aproximaciones de composición semántica de servicios web se dividen en dos: las que tienen en cuanta la calidad del servicio y las que no. Realiza una comparativa de distintas aproximaciones usando como parámetros la calidad del servicio, la semántica, la escalabilidad, la adaptabilidad y el aspecto dinámico entre otros.

En [14] se clasifican las aproximaciones de composición web según el contexto y la tecnología que utilizan. Dentro del contexto diferencia entre la forma de composición (orquestación o coreografía), la semiótica (composición sintáctica o semántica), el mecanismo utilizado (REST o SOAP), el tiempo de diseño (composición manual, semiautomática o automática) y el tiempo de ejecución (composición estática o dinámica). Dentro de la tecnología, clasifica las técnicas de composición en tres tipos: basadas en flujos de trabajo, técnicas dirigidas por modelos y técnicas de planificación de IA.

Según [15] la composición de servicios web tiene cuatro tipos de modelo (orquestación, coreografía, coordinación y modelo de componentes). En este artículo se hace una clasificación de aproximaciones de composición de servicios web automáticas que consiste en *workflow-based*, *model-based*, *mathematics-based* y *AI planning*. Dentro de las aproximaciones basadas en AI planning éstas se dividen en *classical planning*, *neoclassical planning*, *heuristics and control strategies* y otras técnicas.

Por último, tal y como se dice en [16] las técnicas de composición de servicios web pueden ser: basadas en *AI planning* (clásicas, neoclásicas y heurísticas), basadas en matemáticas (grafos, lógica y álgebra de procesos), basadas en modelos (redes de Petri y máquinas de estado finito) y otras (bases de datos y computación evolutiva).

2.3.2 Sistemas existentes

En esta sección se comentan y comparan algunos sistemas existentes que abordar una problemática parecida a la que se plantea en este trabajo. Los sistemas principales que existen son *SSWAP.info* y *mashape*.

SSWAP.info [26] es una aplicación que permite construir flujos semánticos de servicios web directamente de otros sitios web o usando palabras clave para encontrar el servicio deseado. Ofrece funcionalidades similares a las de la interfaz desarrollada en este proyecto aunque con la diferencia de que no permite la configuración de los servicios web utilizados ni la ejecución manual de cada servicio (se ejecuta todo el flujo de principio a fin). Además, tampoco permite la gestión de las descripciones de los servicios.

Mashape [27] es un mercado mundial para consumir, distribuir, administrar y monitorizar APIs públicas y privadas de desarrolladores de todo el mundo. Funciona mas como un gran repositorio de servicios web que como un sistema de composición ya que no permite crear flujos de trabajo con los servicios que utiliza para componer uno nuevo.

2.3.3 El problema de la composición de Servicios Web

Según [10] dado un repositorio de descripciones de servicios, y una consulta con los requisitos del servicio solicitado, en caso de que no se encuentre un servicio que cumpla con los requisitos solicitados, el problema de la composición consiste en encontrar un grafo acíclico dirigido de servicios que se pueden componer para obtener el servicio deseado.

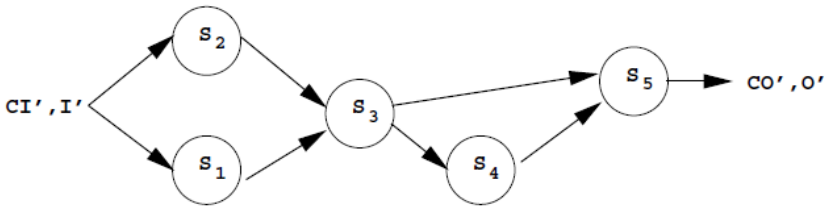


Figura 3: Ejemplo de composición de servicios como un grafo acíclico dirigido

La figura 3 muestra un ejemplo de servicio compuesto de cinco servicios de S_1 a S_5. En la figura, *I'* y *CI'* son los parámetros de entrada y precondiciones de la consulta respectivamente. *O'* y *CO'* son los parámetros de salida y postcondiciones respectivamente. Informalmente, el arco dirigido entre nodos S_i y S_j indica que las salidas de S_i constituyen (algunas de) las entradas de S_j.

Como ejemplo, supongamos que estamos buscando un servicio para comprar un libro y el directorio de servicios contiene los servicios que se muestran en la figura 4. En esta figura también se ven los parámetros de entrada y de salida de la consulta y los servicios.

Service	*Input Parameters*	*Pre-Conditions*	*Output Params*	*Post-Conditions*
Query	*BookTitle,CreditCardNum, AuthorName,CardType*		*ConfNumber*	
GetISBN	*BookName,AuthorName*		*BookISBN*	
GetAvailability	*BookISBN*		*NumAvailable*	*NumAvailable* > 0
AuthorizeCreditCard	*CreditCardNum*		*AuthCode*	*AuthCode* $> 99 \wedge$ *AuthCode* < 1000
PurchaseBook	*BookISBN, NumAvailable, AuthCode*	*NumAvailable* > 0	*ConfNumber*	

Figura 4: Escenario de ejemplo para el problema de la composición.

Supongamos que el repositorio no encuentra un solo servicio que coincida con los criterios de la consulta; entonces se sintetiza un servicio compuesto de entre el conjunto de servicios disponibles en el repositorio. La figura 5 muestra este servicio compuesto.

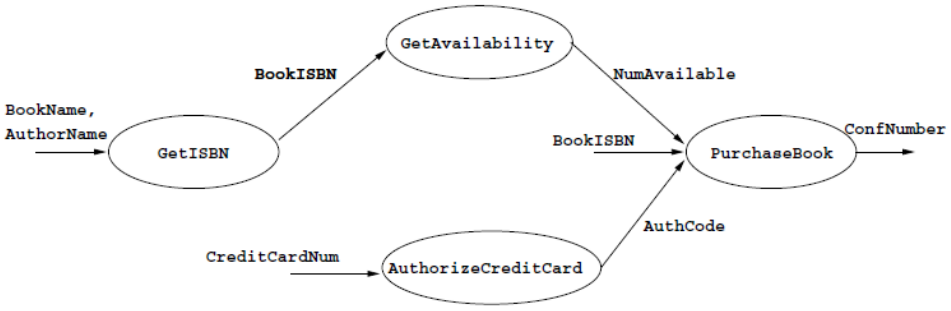
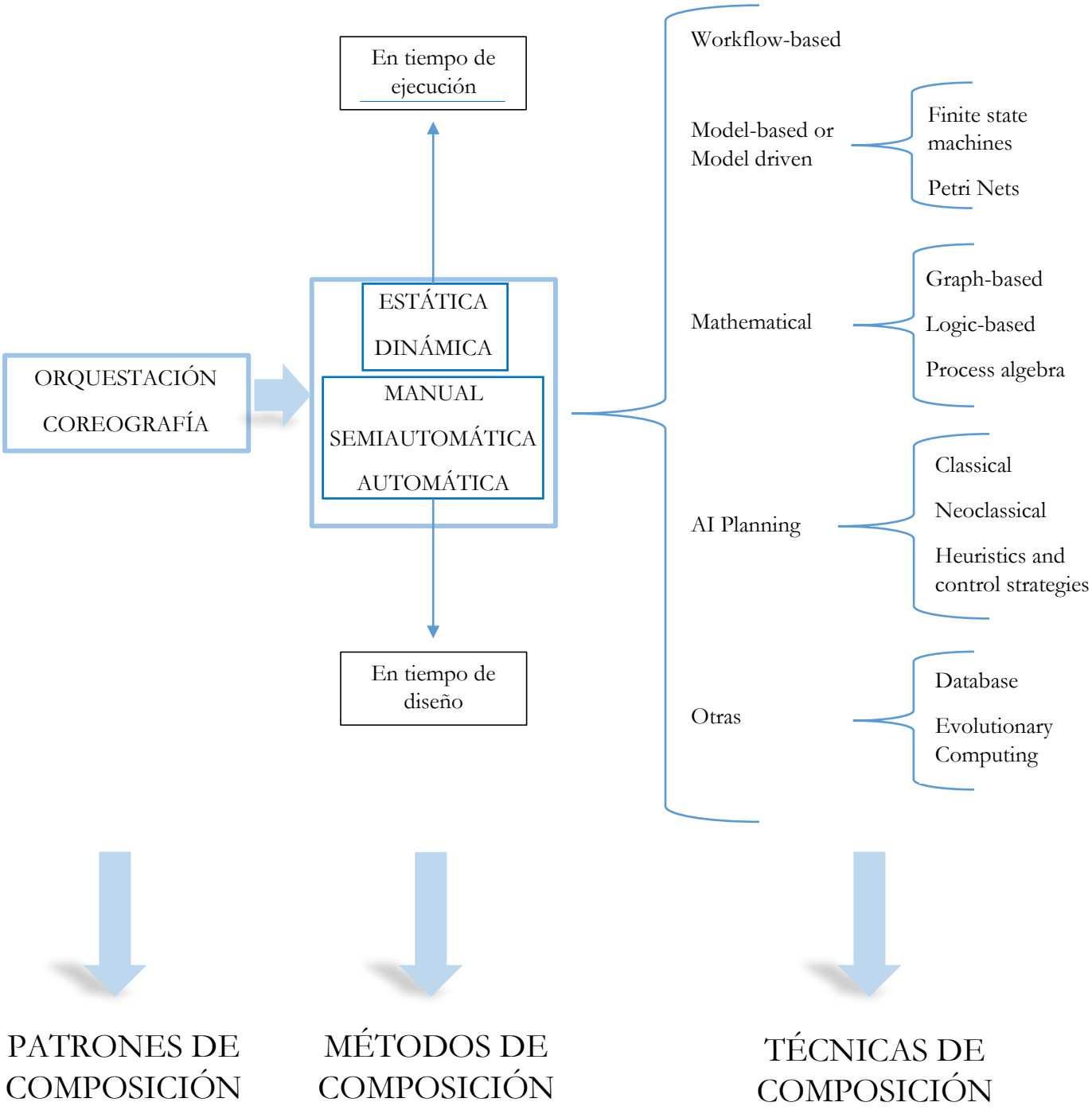


Figura 5: Ejemplo de composición de un servicio

Cada vértice del grafo representa un servicio en la composición. Cada arista de salida de un nodo (servicio) representa las salidas y las postcondiciones producidas por el servicio. Cada arista entrante de un nodo representa las entradas y precondiciones del servicio.

Un servicio en cualquier etapa de la composición puede tener potencialmente como sus entradas todas las salidas de sus predecesores, así como las entradas de la consulta. Los servicios en la primera etapa de la composición solo pueden utilizar las entradas de la consulta. La unión de las salidas generadas por los servicios en la última etapa de la composición debe contener todas las salidas que la consulta requiere que se produzcan. También las postcondiciones de los servicios en cualquier etapa de la composición deben implicar las precondiciones de los servicios en la siguiente etapa.

2.3.4 Taxonomía de la composición de Servicios Web



En la composición de servicios web existen principalmente dos patrones o formas de composición: orquestación y coreografía. A partir de estos patrones se pueden utilizar distintos métodos de composición según el tiempo de diseño o el tiempo de ejecución. Además, existen distintas técnicas para realizar la composición. En este esquema se presentan algunas de las más utilizadas. En concreto, en este trabajo se ha utilizado un método de composición manual basado en flujos de trabajo (Workflow-based).

Para ver con más detalle la explicación de los patrones, métodos y técnicas de composición presentadas en esta taxonomía se recomienda consultar [17].

2.3.5 Composición de Servicios Web REST

La composición de servicios web RESTful es un campo reciente de investigación que todavía no ha sido muy estudiado. Sin embargo, existen algunos trabajos que tratan de abordar esta problemática.

En [18] se presenta una aproximación para componer servicios vinculados con la Web basada en los principios de los datos vinculados y REST. Se propone un método unificado para descubrir tanto las posibilidades de interacción de servicios ofertados como los enlaces semánticos disponibles a otros servicios. El motor de composición se implementa como un cliente genérico que permite explorar la API de un servicio e interactuar con otros servicios para responder al objetivo del usuario.

En [19] se propone un enfoque para la composición de servicios web REST que se basa en la descripción de servicios REST centrados en hipermedia. Usa el lenguaje ReLL (Resource Linking Language) y utiliza la técnica de Redes de Petri como mecanismo para describir la navegación máquina-cliente.

El trabajo presentado en [20] propone un modelo formal para la descripción individual de los servicios web y para la composición automática. Se usa un sistema de transición de estados basado en la técnica Situation Calculus (pertenece a las técnicas de IA) para automatizar el proceso de composición de servicios web RESTful.

También existen aproximaciones como [21] que proponen el uso del lenguaje BPEL para extensiones REST. El objetivo de este trabajo es permitir la composición tanto de servicios web RESTful como la de servicios web tradicionales a partir del mismo lenguaje de composición de servicios orientado a procesos. Además, se muestra como publicar un proceso BPEL como un servicio web RESTful exponiendo las partes seleccionadas de su estado de ejecución usando las primitivas de interacción REST.

Otro trabajo que aborda la composición de servicios web RESTful es [22] en el que se presenta una aproximación ligera para la composición de servicios web RESTful y flujos de trabajo colaborativos que usa Bite, un lenguaje de composición ligero y extensible que permite la creación de flujos de trabajo de escala Web y usa los servicios RESTful como sus principales entidades componibles.

Por último, en [23] se presenta un trabajo que utiliza JOpera (lenguaje de composición visual) para la composición de servicios web RESTful. En este trabajo se incluye un caso de estudio con el que se muestra, usando JOpera, como construir una aplicación compuesta fuera de las bien conocidas y actuales APIs de servicios RESTful.

2.4 Conclusiones

REST es un estilo arquitectónico ligero y simple que nos permite aprovechar las características de la Web que la han hecho tan exitosa. Debido a las ventajas que ofrece como una mayor escalabilidad o el uso de interfaces uniformes ha aumentado su uso en detrimento del protocolo SOAP más pesado y complejo.

Los servicios web proporcionan un mecanismo mediante el que se puede estandarizar la comunicación entre diferentes sistemas y aplicaciones gracias a que utilizan un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Se caracterizan por ser débilmente acoplados, por proporcionar interoperabilidad y por tener un desarrollo relativamente fácil y rápido.

Además, gracias a la aparición de REST es posible utilizar servicios web RESTful que aprovechen las características de la web. Estos servicios se presentan como alternativa a SOAP ya que son ligeros y sin estado.

La composición de servicios web RESTful es un campo reciente que todavía no ha sido ampliamente estudiado aunque existen algunos trabajos sobre composición de servicios RESTful que utilizan técnicas de composición ya conocidas.

3 DESARROLLO DE LA INTERFAZ

En este capítulo se presenta el desarrollo de la interfaz implementada. En primer lugar, se analizan los requisitos que debe cumplir la interfaz para alcanzar los objetivos establecidos. A continuación, se comenta el diseño y la implementación realizados para desarrollar la interfaz. Por último, se especifican las pruebas realizadas y se explica un escenario de uso para mostrar el correcto funcionamiento de la aplicación.

3.1 Definición de requisitos

En este apartado se definen los requisitos que tiene cumplir la interfaz web que se quiere desarrollar para conseguir los objetivos establecidos en el apartado 1.3. La interfaz está formada por los módulos de gestión de descripciones y de composición de servicios por lo que en las siguientes secciones se especifican los requisitos propios de cada uno de ellos. Se parte del contexto de que existe un repositorio de servicios web RESTful disponible. Además, los servicios se consideran recursos y se asume que se identifican mediante URIs.

3.1.1 Requisitos del módulo de gestión de descripciones de servicios web

El módulo de gestión de descripciones de servicios web REST tiene como objetivo permitir que el usuario pueda habilitar servicios en la interfaz mediante sus descripciones. Esto significa que el usuario puede utilizar un servicio en la interfaz añadiendo una descripción del mismo. Además la gestión de descripciones incluye tareas básicas de gestión como son la eliminación, modificación y visualización de descripciones. Teniendo en cuenta todo, esto los requisitos específicos de este módulo son:

- **Añadir descripción:** la interfaz permitirá que el usuario pueda añadir una descripción de un servicio no compuesto para que éste se pueda utilizar. Una vez añadida la descripción, el servicio descrito pasará a estar disponible para su uso en el módulo de composición.

- **Eliminar descripción:** la interfaz permitirá que el usuario pueda eliminar una descripción de un servicio, tanto si es compuesto como si no, para que éste deje de estar disponible y ya no se pueda utilizar. Una vez eliminada la descripción, el servicio descrito dejará de estar disponible para su uso en el módulo de composición.

- **Modificar descripción:** la interfaz permitirá que el usuario pueda modificar una descripción de un servicio no compuesto. Esto puede ser necesario en el caso de que cambie el servicio y se necesite añadir o quitar datos de su descripción.

- **Visualizar descripción:** la interfaz permitirá que el usuario pueda visualizar una descripción de un servicio no compuesto. De esta manera el usuario podrá ver toda la información relativa a ese servicio.

- **Listar descripciones:** la interfaz permitirá listar las descripciones añadidas tanto de los servicios simples como de los compuestos.

Referente a los servicios compuestos, simplemente se podrán listar y eliminar sus descripciones ya que, como se verá más adelante, sus descripciones se generan a partir de los servicios que los componen. Por lo tanto, las tareas de añadir un servicio compuesto o visualizarlo serán realizadas en el módulo de composición de servicios.

3.1.2 Requisitos del módulo de composición de servicios web REST

El módulo de composición de servicios web REST tiene como objetivo permitir que el usuario pueda componer un nuevo servicio web a partir de otros servicios descritos en la interfaz. Esto significa que el usuario puede utilizar los servicios descritos para crear un nuevo servicio web compuesto representado por un grafo acíclico dirigido. Este nuevo servicio se podrá ejecutar para ver su funcionamiento y se podrá guardar para poder recuperarlo posteriormente. Además, cada servicio que compone el servicio compuesto podrá ser configurado y ejecutado individualmente si así se desea. Teniendo en cuenta todo esto los requisitos específicos de este módulo son:

- **Listar servicios simples:** la interfaz deberá listar los servicios simples descritos en el módulo de gestión de descripciones para que se puedan utilizar en el proceso de composición de un nuevo servicio.

- **Listar servicios compuestos:** la interfaz deberá listar los servicios compuestos que se hayan guardado en un proceso de composición.

- **Añadir servicio a la composición:** la interfaz permitirá que el usuario seleccione un servicio y lo añada al grafo de composición para que pueda ser usado.

- **Eliminar servicio de la composición:** la interfaz permitirá eliminar un servicio del grafo de composición de tal manera que ese servicio ya no formará parte de la composición.

- **Borrar grafo de composición:** la interfaz permitirá borrar todo el grafo de composición para poder crear uno nuevo.

- **Configurar servicio:** la interfaz deberá permitir la configuración de un servicio para indicar los valores de los parámetros y el origen de esos valores. También deberá permitir indicar si el servicio es iterable, la frecuencia de iteración y la fuente de resultados a partir de la cúal se va a iterar.

- **Visualizar información del servicio:** la interfaz deberá permitir visualizar la información de cada servicio simple añadido al grafo de composición.

- **Ejecutar servicio:** una vez configurado, la interfaz permitirá ejecutar un servicio para ver sus resultados.

- **Ver resultados de ejecución del servicio:** una vez ejecutado, la interfaz mostrará los resultados del servicio obtenidos tras su ejecución.

- **Ejecutar composición:** la interfaz permitirá ejecutar todo el grafo de composición que representa el servicio compuesto para ver su resultado. Para ello, los servicios que forman parte del grafo de composición deberán estar configurados previamente si así lo requieren.

- **Guardar composición:** la interfaz deberá permitir guardar el servicio compuesto para que éste esté disponible para ser usado en otra composición.

- **Iterar servicios:** la interfaz permitirá iterar un servicio simple, es decir, que se ejecute un número determinado de veces en función de la fuente de resultados a partir de la que se va a iterar. Por cada iteración obtendrá unos datos de entrada a partir de los cuáles producirá un resultado que se almacenará.

- **Conectar servicios:** la interfaz permitirá conectar los servicios mediante flechas para poder construir el grafo de composición.

- **Etiquetar flechas:** la interfaz permitirá etiquetar las flechas del grafo de composición para poner nombres representativos de los resultados del servicio del que parte la flecha.

3.2 Diseño

Como se ha visto hasta ahora, la aplicación se ha dividido en dos módulos diferentes. El primer módulo es el de la gestión de descripciones de servicios web que se encarga de habilitar servicios mediante sus descripciones para que se puedan utilizar en el proceso de composición. El segundo módulo es de composición de servicios y es el que permite sintetizar un nuevo servicio a partir de algunos de los servicios existentes. Este proceso de composición será manual, ya que como se ha visto en el estado del arte la composición automática es muy compleja y la mayoría de enfoques proponen soluciones manuales o semiautomáticas. Además, de las técnicas mostradas en la taxonomía de la composición de servicios web vista en el estado del arte se ha utilizado una basada en flujos de trabajo, ya que permite la ejecución secuencial de los servicios.

Tal y como se muestra en el apartado 1.6, ambos módulos se comunican con el servidor a través de una API REST. Además los servicios web que se utilizan también son RESTful. Por tanto, dado que lo que se pretende es realizar la composición de servicios web RESTful así como la gestión de sus descripciones, se deben cumplir una serie de principios de diseño propios de REST.

En primer lugar, la identificación única de los recursos se hará mediante el uso de URI.

En segundo lugar la interfaz de la aplicación seguirá el principio de REST Interfaz uniforme para manipular los recursos a través de sus representaciones. Los distintos tipos de operaciones se realizarán usando los verbos HTTP: GET, POST, PUT y DELETE.

En tercer lugar, se usarán mensajes autodescriptivos. REST dicta que los mensajes HTTP deberían ser tan descriptivos como sea posible para que sea posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario. HTTP logra esto por medio del uso de varios métodos estándares, muchos encabezamientos y un mecanismo de direccionamiento. HTTP define una serie de códigos de mensaje para indicar las diferentes situaciones en las que se puede encontrar el procesamiento de una operación.

En cuarto lugar se debe utilizar hipermedia como el motor de estado de la aplicación (HATEOAS). El estado se incluye en el recurso. Mediante enlaces se definen relaciones entre los recursos y las transiciones de estado válidas de la interacción del servicio.

En quinto lugar la comunicación será sin estado, es decir, sin establecer ninguna sesión entre el cliente y el servidor. El estado se mantendrá en el cliente, el servidor no guardará el estado de la comunicación.

Además de cumplir con los principios de diseño de REST, el intercambio de información entre los módulos, el servidor y los servicios se realiza utilizando el formato JSON que es más ligero y simple de utilizar que XML.

A parte de lo que se acaba de mencionar, cada módulo deberá seguir el diseño que se detalla a continuación.

3.2.1 Módulo de gestión de descripciones de servicios web

Como se ha visto anteriormente, el módulo de gestión de descripciones de servicios se encarga de gestionar las decripciones de los servicios que se desea que estén disponibles para el proceso de composición. Este módulo debe cumplir con los requisitos especificados en el apartado 3.1.1 y para ello se han realizado las siguientes tareas.

Alta de descripciones

Para que un servicio web esté disponible en el módulo de composición, se necesita añadir una descripción de este servicio. Para ello, en el módulo de gestión de descripciones, se debe rellenar un formulario (ver figura 6) que contiene los campos necesarios para crear la descripción del servicio que se quiere habilitar. Estos campos son:

- **Nombre:** nombre del servicio que se quiere describir. No tiene porque coincidir con el nombre real, pudiéndose escribir un nombre que resulte más representativo del servicio. Es un campo obligatorio, ya que todos lo servicios deben tener un nombre que los represente cuando se cree el grafo de composición.

- **Method:** indica el método HTTP que se utiliza para llamar al servicio.
- **Documentation:** es la url del servicio en la que se encuentra su documentación.
- **Endpoint:** es la url del servicio que lo identifica como recurso. Se usa para acceder al recurso mediante la correspondiente llamada HTTP. Este campo es obligatorio pues sin endpoint no se puede acceder al servicio.
- **Description:** descripción de lo que hace el servicio.
- **Parameters:** posibles párametros del servicio que se tienen que rellenar para obtener un resultado correcto. Se pueden añadir tantos parámetros como sean necesarios. Por cada parámetro hay una serie de campos que hay que rellenar:
    - **Name:** nombre del parámetro. Debe coincidir con el nombre real del parámetro correspondiente del servicio.
    - **Required:** indica si el parámetro es obligatorio o no.
    - **Place:** indica el lugar en el que se envía el parámetro: path, parámetro de url o payload.
    - **Type:** indica el tipo de dato del parámetro.
    - **Default value:** indica el posible valor por defecto que puede tomar el parámetro si no se rellena.
    - **Valid options:** indica las posibles opciones que acepta el parámetro.
    - **Description:** descripción de lo que significa el parámetro.

Téngase en cuenta que la información de todos estos campos se obtiene del servicio, y por tanto debe coincidir para que la descripción sea correcta y se pueda acceder al servicio de forma adecuada.

Una vez obtenidos los datos del formulario se construye un JSON con la información y se envía al servidor mediante una llamada HTTP PUT. La API REST del servidor recibe los datos, hace las comprobaciones pertinentes y almacena la descripción. En el caso de que hubiera algún tipo de error el servidor envía un mesaje de error que se recupera y se muestra al usuario.

Modificación de descripciones

Si se quiere modificar algún campo de la descripción de un servicio se deberá cambiar lo que se desee mediante un formulario igual al del alta de descripciones. La única diferencia es que en el formulario de modificación (ver figura 8) se recuperan los valores almacenados haciendo una solicitud HTTP GET a la API REST del servidor con el identificador de la descripción. Incluso la llamada HTTP que se hace al servidor es la misma ya que si recibe una solicitud HTTP PUT con una descripción que ya existe simplemente la actualiza. Al igual que en el alta de descripciones, en el caso de que hubiera algún tipo de error el servidor envía un mesaje de error que se recupera y se muestra al usuario.

Eliminación de descripciones

Para el proceso de eliminación de una descripción se hace una llamada HTTP DELETE a la API REST del servidor en la que se le envía el identificador de la descripción que se quiere eliminar. Por precaución antes de hacer la solicitud de borrado se pide confirmación al usuario.

Visualización de descripciones

En el caso de la visualización de una descripción (ver figura 9), al igual que en la modificación, se recuperan los valores almacenados haciendo solicitud HTTP GET a la API REST del servidor con el identificador de la descripción que se quiere visualizar.

Listado de descripciones

Para ver el listado de descripciones de los servicios (ver figura 7) se hace una solicitud HTTP GET al servidor sin ningún identificador de descripción, y éste nos devuelve el listado completo de las descripciones almacenadas.

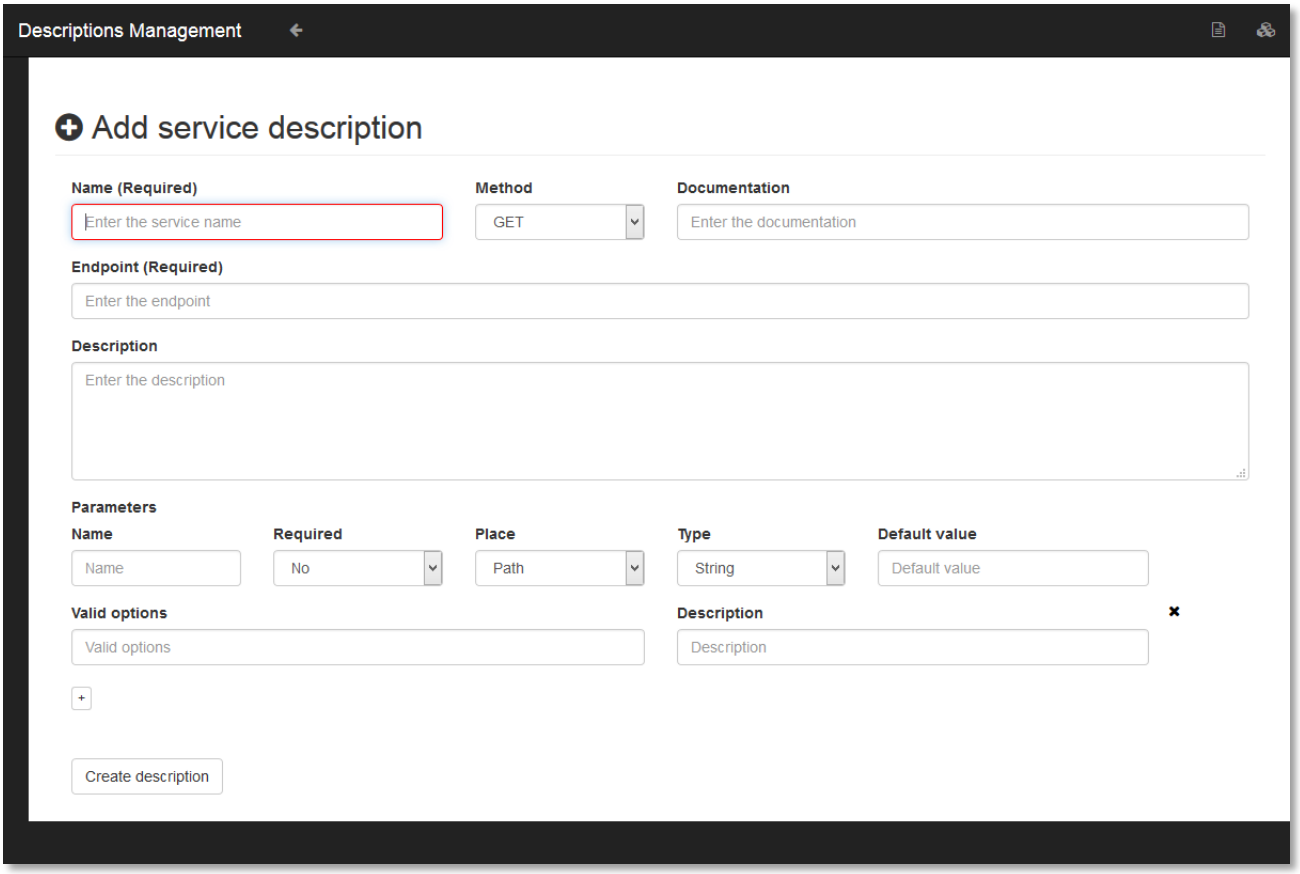


Figura 6: Formulario para añadir la descripción de un servicio web

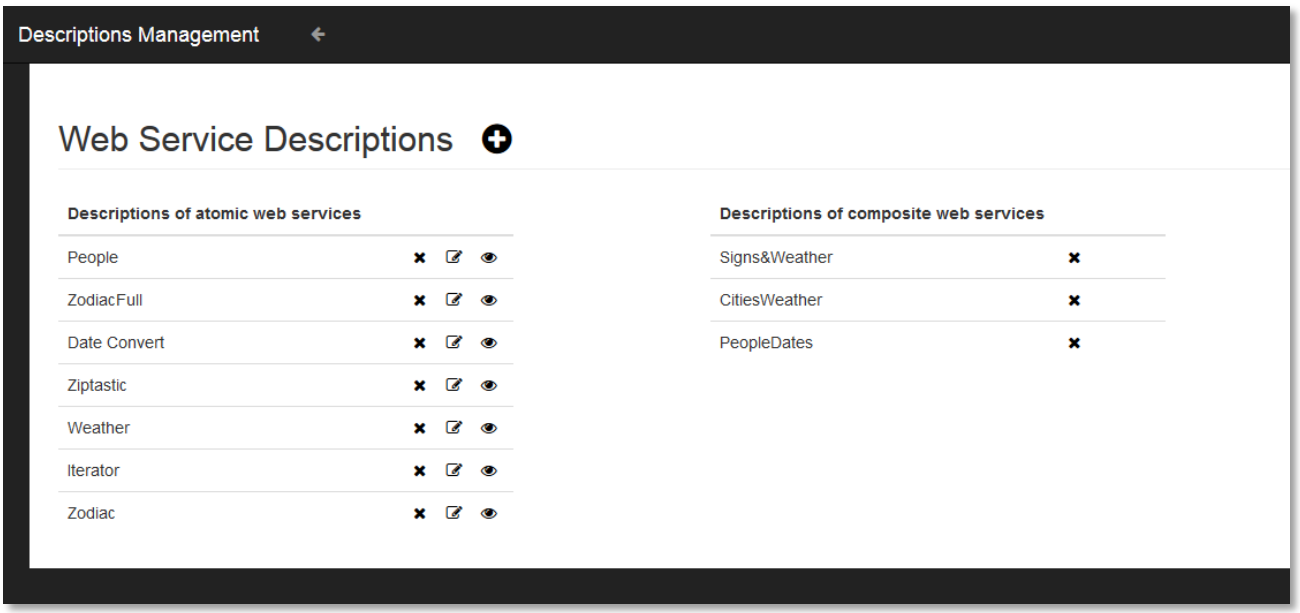


Figura 7: Listado de descripciones de servicios

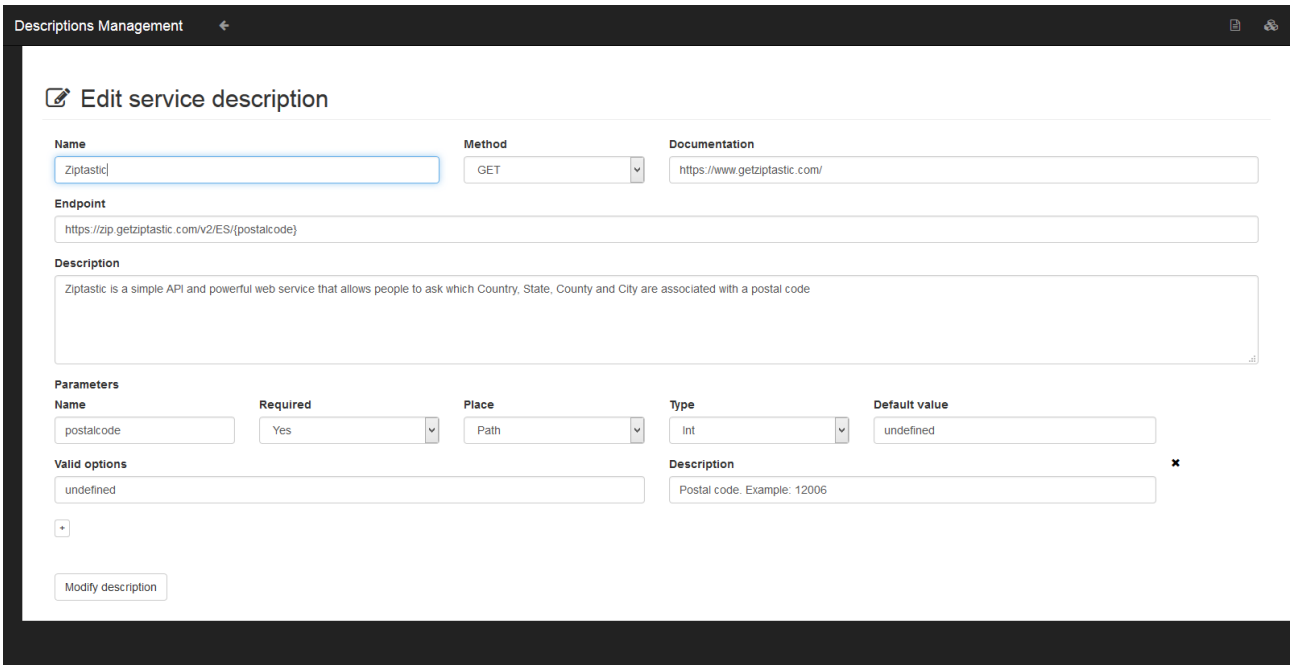


Figura 8: Formulario para modificar la descripción de un servicio

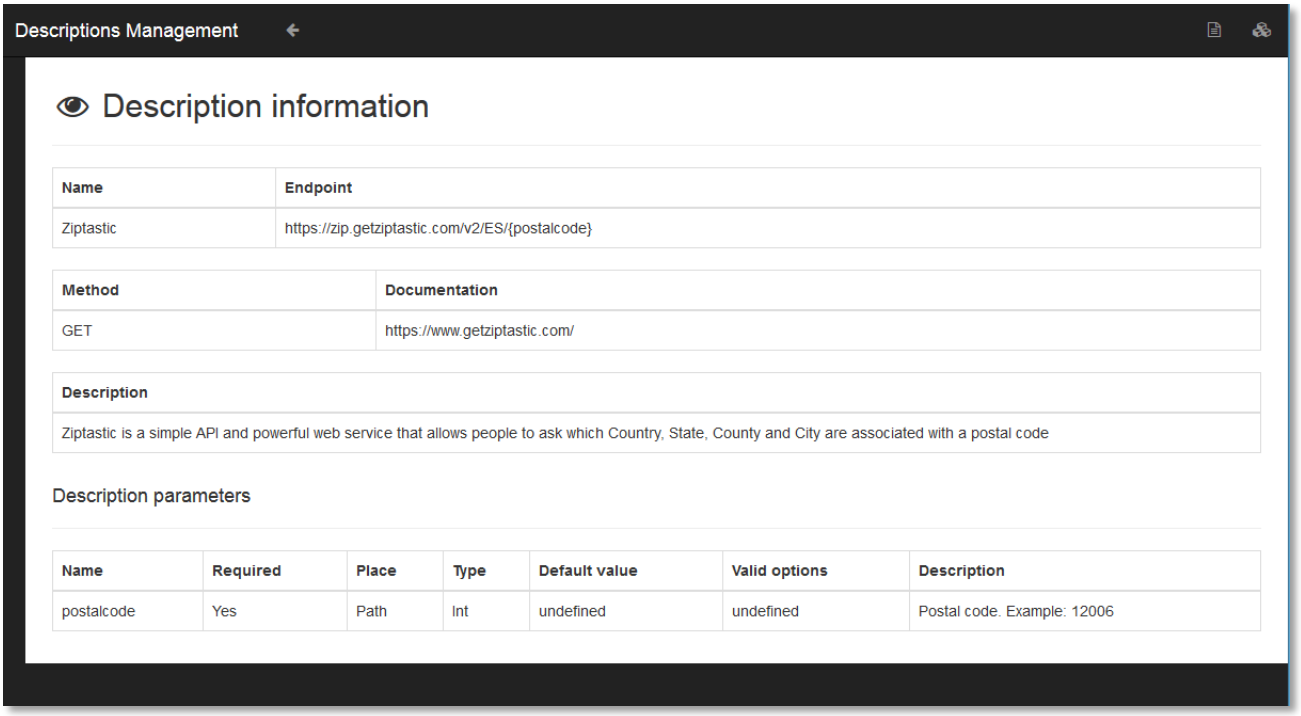


Figura 9: Visualización de la descripción de un servicio

3.2.2 Módulo de composición de servicios web

Como ya se ha visto anteriormente, el módulo de composición de servicios web REST tiene por objetivo permitir la composición de un nuevo servicio mediante la reutilización de los servicios descritos en el módulo de gestión de descripciones. Este módulo debe cumplir con los requisitos especificados en el apartado 3.1.2 por lo que para cada uno de ellos se explican los detalles del diseño realizado.

Listado de servicios

Para poder elegir los servicios que se quieren utilizar en el proceso de composición se muestra, por un lado, una lista con los nombres de todos los servicios que han sido descritos en el módulo de gestión de descripciones (servicios simples) y por el otro, una lista con todos los servicios compuestos guardados que se pueden utilizar también en un proceso de composición (ver figura 10). Para obtener estas listas se hacen dos solicitudes HTTP GET a la API REST del servidor (una petición por cada lista) y ésta nos devuelve los JSONs correspondientes con la información.

Área de composición

Para crear el grafo de composición de servicios se dispone de un área de composición (ver figura 10) en la que se pueden ir incorporando los servicios necesarios para realizar este proceso. Esta área de composición representa al contenedor donde se almacenan todos los elementos de grafo y se ha implementado usando la librería JavaScript Cytoscape.js.

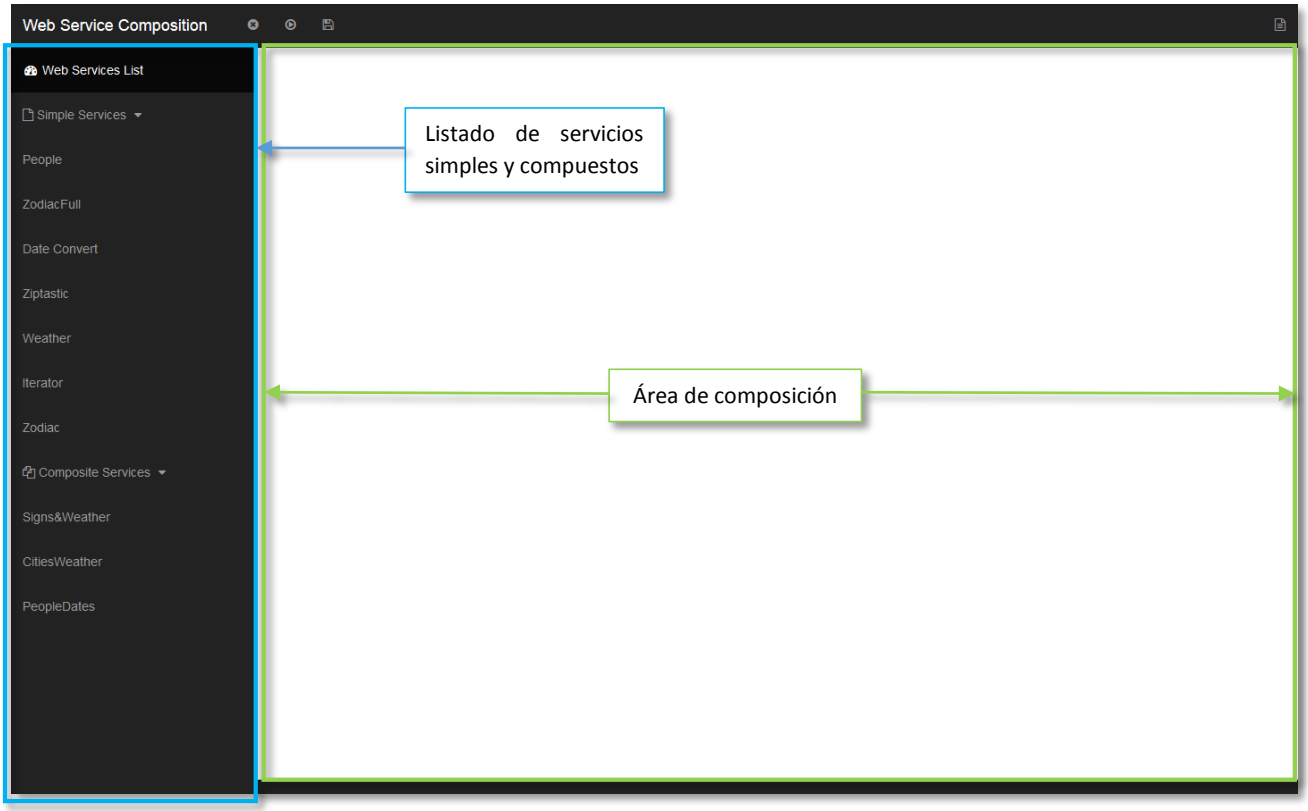


Figura 10: Detalle del listado de servicios y área de composición

Creación del grafo de composición

Para componer un servicio compuesto se debe crear un grafo de composición (ver figura 11) donde cada nodo del grafo representa a un servicio simple. Los nodos se añaden arrastrando y soltando un nombre de un servicio de la lista de servicios al área de composición. Esta funcionalidad se ha implementado utilizando la API de HTML5 Drag & Drop (arrastrar y soltar). Las flechas del grafo se crean con la librería Cytoscape.js y representan la información que sale de un nodo y que llega a otro. Además, mediante la librería Cytoscape.js todos los elementos que se añaden al área de composición pasan a formar parte inmediatamente del grafo de composición. El proceso de creación de un servicio compuesto se ha diseñado de manera que quede claro que servicios son necesarios y en que orden, por ello el proceso de composición se representa como un grafo acíclico dirigido en el que queda claro que servicios se tienen que ejecutar antes y en que orden se pasa la información de un servicio a otro. Cuando se añade un servicio al área de composición se hace una petición HTTP GET a la API REST del servidor para

recuperar su descripción. La información de esta descripción se almacena en un diccionario de descripciones para usarla posteriormente cuando se quiera configurar o ejecutar el servicio.

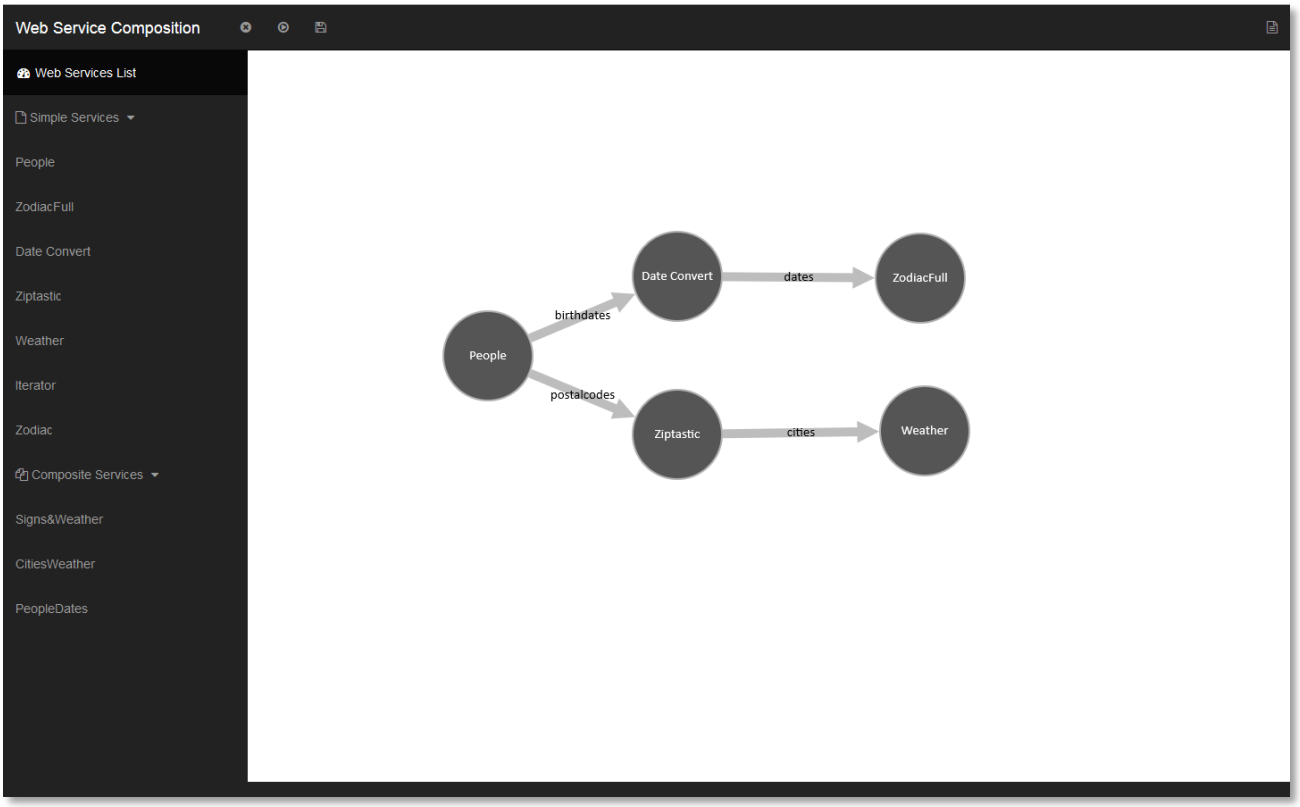


Figura 11: Grafo de composición de un servicio compuesto

Menú contextual

Para que al usuario le resulta más cómo trabajar con el grafo se ha diseñado el uso de un menú contextual (ver figura 12) que permite acceder a diferentes funcionalidades. Si el elemento es un nodo se puede eliminar ese nodo, configurarlo o ejecutarlo. Si se trata de una flecha se puede eliminar la flecha o reetiquetarla. Este menú se ha implementado usando Cytoscape-cxtmenu.js, un plugin para la librería Cytoscape.js que crea un widget que permite usar un menú contextual circular sobre el elemento del grafo sobre el que hagamos click.

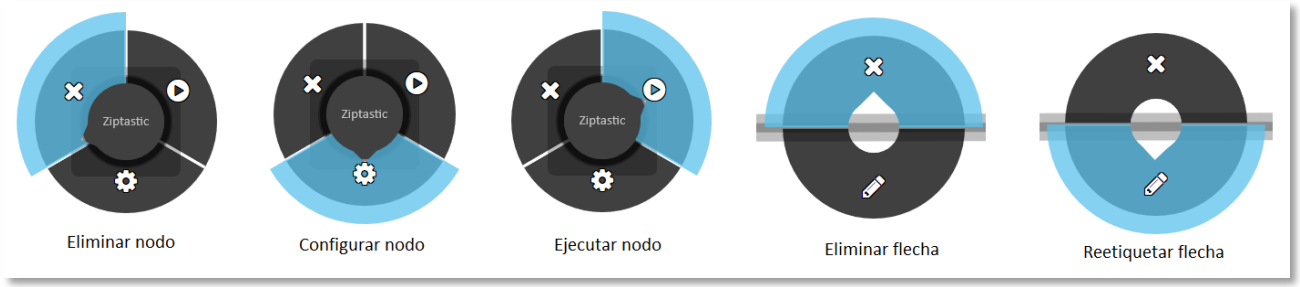


Figura 12: Funciones menú contextual

Eliminación de elementos del grafo de composición

El proceso de eliminación de un elemento del grafo de composición se ha diseñado de dos maneras: o bien se pueden eliminar todos los elementos del grafo pulsando el botón de "Delete graph", o bien se puede seleccionar la opción correspondiente del menú contextual que aparece al hacer click con el botón derecho del ratón encima del elemento a eliminar. Ambas maneras se han implementado usando las funciones proporcionadas por la librería Cytoscape.js de tal manera que al realizar esta acción se eliminan los elementos del objeto que representa el grafo.

Visualización de información del servicio

Como se ha explicado anteriormente, cuando se añade un servicio al grafo de composición se recupera del servidor su descripción y se guarda. A partir de la descripción del servicio almacenada se muestra la información de ese servicio en forma de tabla. Esta información se puede ver en la pestaña "Service information" (ver figura 13) que aparece al seleccionar la opción de configuración del menú contextual.

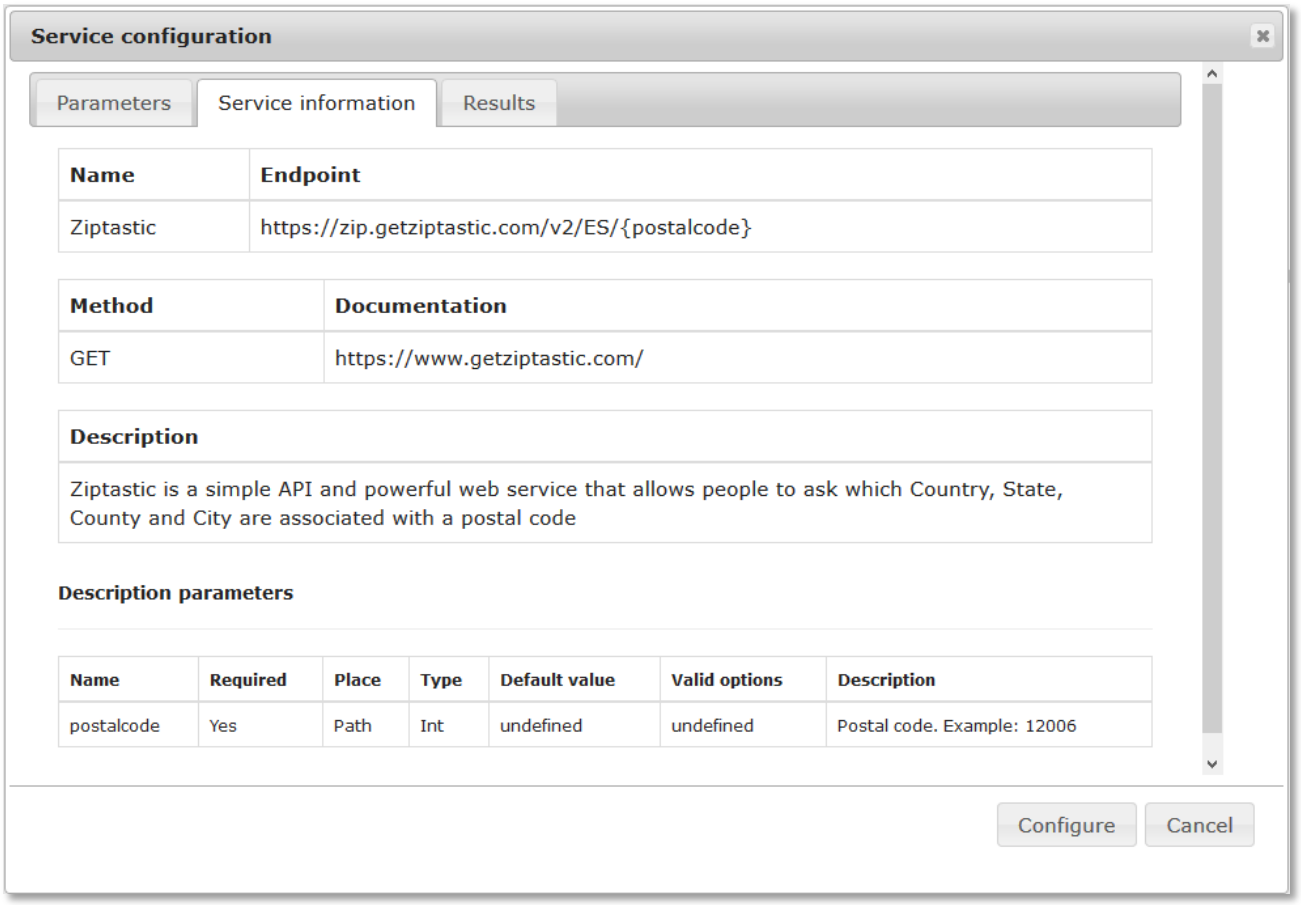


Figura 13: Pestaña información del servicio

Configuración de un servicio

El proceso de configuración del servicio consiste en completar un formulario de configuración (ver figura 14) y almacenar los datos introducidos. Estos datos se almacenan internamente en un diccionario de configuraciones que tiene por claves el id de cada nodo del grafo y como valor la infomación introducida en el formulario. Para mostrar este formulario se recupera la información necesaria del diccionario de descripciones donde está almacenada la descripción del servicio y se crea dinámicamente. En el formulario se muestran todos los parámetros que tiene el servicio para que se indique el valor que toman y de dónde lo toman. El origen de los datos puede ser manual (se introduce directamente el valor) o puede proceder de los resultados de ejecución de uno de los nodos conectados al nodo que se quiere ejecutar. En este último caso se debe indicar mediante una expresión JSON el valor que tomará el parámetro (posteriormente, en el manual de usuario se podrá ver con más detalle cómo se usa cada parte de la interfaz diseñada). Puede ocurrir que un servicio no tenga parámetros y por tanto no sea necesario configurarlo; en ese caso se muestra al usuario un mensaje informativo. También se muestran unas opciones relacionadas con el proceso de iteración que se explican más adelante.

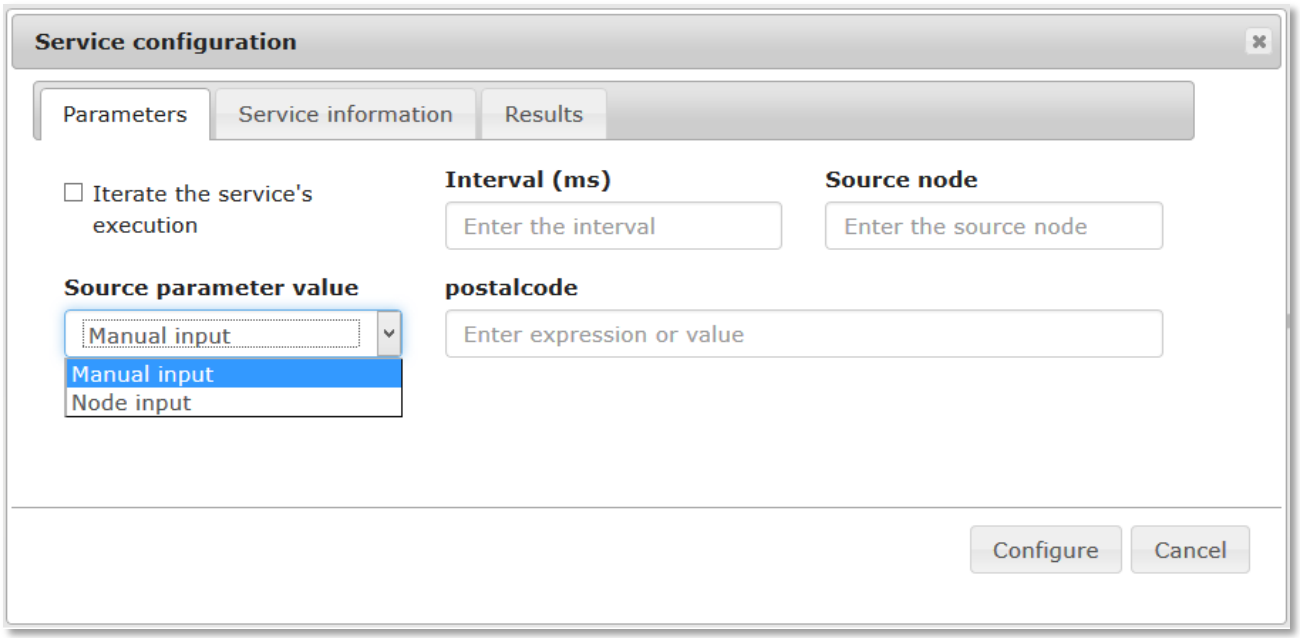


Figura 14: Formulario de configuración

Ejecución de un servicio

Cuando se quiere ejecutar un servicio se necesita que éste haya sido previamente configurado si tiene parámetros. A partir de la descripción del servicio se obtiene el endpoint y el método que se tiene que utilizar para hacer la petición correspondiente al servicio. Si el servicio tiene parámetros se obtiene su configuración del diccionario de configuraciones y se colocan estos parámetros en el lugar que corresponda (según se indique en la descripción del servicio). Si algún parámetro debe ir en el path o como parámetro del endpoint, se construye el nuevo endpoint con el valor de los parámetros necesarios. En el caso de que algún parámetro vaya en el payload (cuerpo de la petición) se construye un JSON con ellos y se envía junto con la petición HTTP. Una vez recogidos y colocados los posibles parámetros se hace la correspondiente solicitud HTTP al servicio y se almacenan los resultados en un diccionario de resultados.

Visualización de los resultados de ejecución de un servicio

Una vez se ha ejecutado un servicio se pueden ver sus resultados de ejecución en la pestaña "Results" (ver figura 15) que aparece al seleccionar la opción configuración del menú contextual. El formato utilizado para el intercambio de datos es el formato JSON y los resultados que se obtienen se presentan siguiendo ese formato. Hay que tener en cuenta que en los resultados se muestra exactamente la respuesta del servicio a partir de los datos que se le han pasado lo que significa que si los datos introducidos no son válidos en los resultados se mostrarán los mensajes de error del servicio.

Iteración de un servicio

Normalmente suele ocurrir que se quiere ejecutar un servicio para una lista de datos. Esto quiere decir que el servicio recibe una lista de datos y por cada dato debe ser ejecutado. A esto se le llama iteración de un servicio y consiste en ejecutar el servicio tantas veces como sean necesarias para obtener un resultado por cada dato de la lista recibida. Como se ha visto en el proceso de configuración, se puede indicar si el servicio es iterable y si lo es se tiene que indicar la frecuencia de ejecución (el tiempo que se espera entre llamada y llamada a un servicio para evitar ser bloqueado) y el nodo fuente del que se van a tomar los resultados a partir de los que se quiere iterar. Esto se ha implementado usando la misma función de ejecutar, que en el caso de que el servicio sea iterable ejecutará tantas veces como sea necesario el servicio y se esperará el tiempo indicado entre ejecución y ejecución.

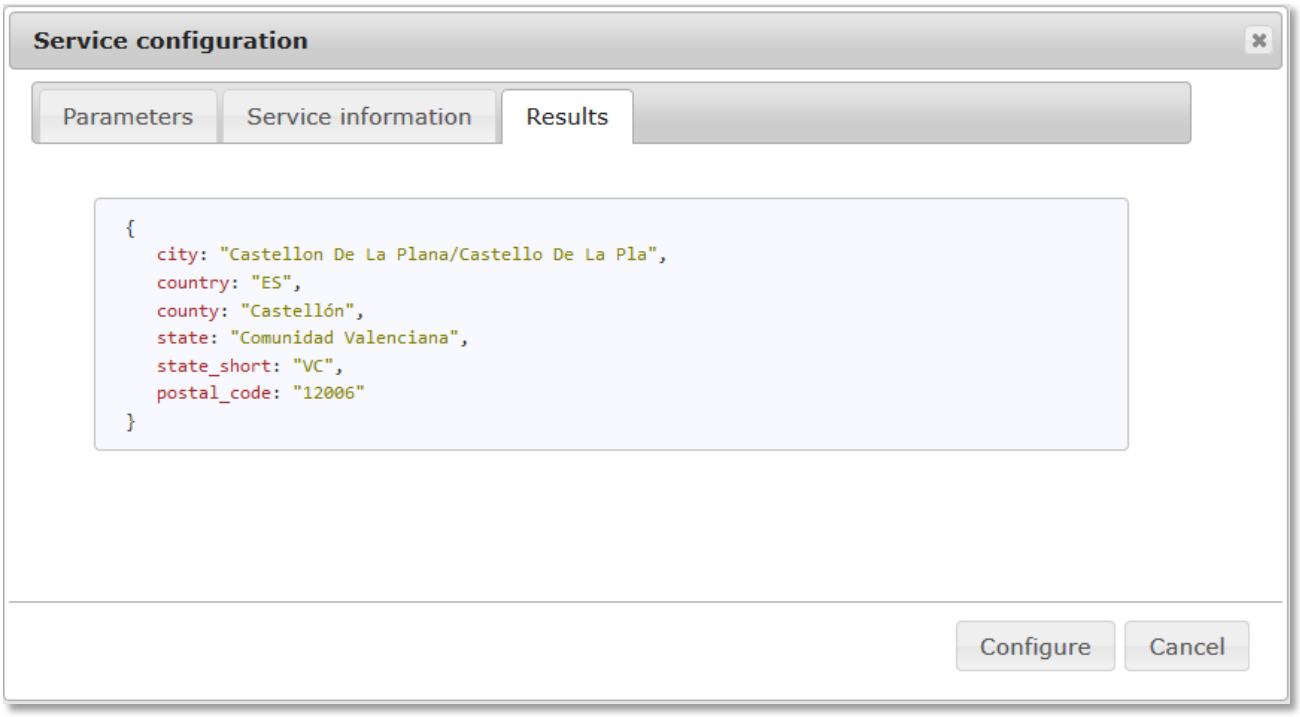


Figura 15: Visualización de los resultados de ejecución de un servicio

Ejecución de una composición

La ejecución de un servicio compuesto implica recorrer el grafo para conocer el orden de ejecución de los nodos. Esto es importante ya que es necesario ejecutar primero los nodos predecesores de otros nodos para que éstos últimos puedan obtener los datos necesarios para su ejecución. Una vez establecido el orden de ejecución se van ejecutando uno a uno todos los servicios usando para ello la misma función que se utiliza para ejecutar un único nodo. Los resultados obtenidos en la ejecución de cada nodo se van almacenando en un diccionario de resultados para que puedan ser accedidos por los nodos que los necesiten.

Guardar una composición

El proceso de guardar una composición consiste en almacenar en un JSON los nodos y arcos del grafo de composición. Por cada nodo se almacena su identificador y su nombre y por cada arco su origen y su destino. Después de construir el JSON con la información del grafo éste se envía al servidor para ser almacenado mediante una petición HTTP PUT. Una vez almacenado el servicio compuesto está disponible para ser usado.

Recuperación de un servicio compuesto

Recuperar un servicio compuesto consiste en hacer una petición HTTP GET con el identificador del servicio a la API REST del servidor. El servidor nos devuelve un JSON con la información del grafo de composición del servicio compuesto. A partir de este JSON se crea el grafo de composición del servicio añadiendo los nodos y los arcos del grafo al área de composición.

3.3 Implementación

Una vez visto el diseño de cada una de las funcionalidades de la interfaz se explica a continuación cómo se ha realizado la implementación de ambos módulos.

En primer lugar, las peticiones a la API del servidor se realizan mediante llamadas AJAX con XMLHttpRequest. Este tipo de llamadas se realizan principalmente en el módulo de gestión de descripciones para realizar las tareas de crear, modificar, eliminar, visualizar y listar descripciones. En el módulo de composición también se utiliza pero únicamente para recuperar las descripciones de los servicios que se añaden al área de composición y cuando se quiere guardar un servicio compuesto.

En segundo lugar, la implementación del grafo de composición se hace usando la librería JavaScript para grafos Cytoscape.js [25]. El grafo de composición se crea como un objeto que representa un contenedor al que se le pueden ir añadiendo nodos y flechas. En este objeto también se especifican detalles estéticos mediante código CSS. A continuación se muestra un ejemplo:

```
cy = cytoscape({
        container: document.getElementById('cy'),
        zoomingEnabled: false,
        panningEnabled: true,
        boxSelectionEnabled: true,

        style: [
            {
                selector:'core',
                css: {
                    'selection-box-color':"#aaaeff",
                    'selection-box-border-color':"#8BB0D0",
                    'selection-box-opacity':"0.5"}
            },
            {
                selector: 'node',
                css: {
                    'content': 'data(name)',
                    'text-valign': 'center',
                    'text-halign': 'center',
                    'width': 125,
                    'height': 125,
                    'background-color': '#555',
                    'border-width': 2,
                    'border-color': '#AFAFAF',
                    'font-family': 'Calibri',
                    'font-size': 15,
                    'color': '#FFF'
                }
            },
            {
                selector:'node:selected',
                css: {
                    'border-width':"6px",
                    'border-color':"#AAD8FF",
                    'border-opacity':"0.5",
                    'background-color':"#77828C",
                    'text-outline-color':"#77828C"}
            },
            {
                selector: 'edge',
                css: {
                      'content': 'data(label)',
                    'target-arrow-shape': 'triangle',
                    'target-arrow-color': '#BDBDBD',
                    'line-color': '#BDBDBD',
                    'width': '10px',
                      'font-family': 'Calibri',
                    'font-size': 15,
                }
            },
            {
                selector: ':selected',
                css: {
```

```
                'line-color': 'black',
                'target-arrow-color': 'black',
                'source-arrow-color': 'black'
            }
        }
    ],

    layout: {
        name: 'grid',
        padding: 5
    }
});
```

Además, se utilizan varias funciones propias de esta librería que se detallan a continuación:

- **cy.add():** se utiliza para añadir elementos (nodos y arcos) al grafo.

```
cy.add({
        group: "nodes",
        data: {label: 'nodo1' , name: 'nodo1'},
        position: { x: 500 , y: 500}

cy.add([ { group: "edges", data: { source: 'n0', target: 'n1', label: 'flecha1'}}]);
```

- **cy.elements():** se usa para obtener una lista de los elementos que contiene el grafo.

```
var eles = cy.elements()
```

- **cy.remove():** sirve para eliminar uno o varios elemento del grafo.

```
this.remove();

var eles = cy.elements();
eles.remove();
```

- **cy.edges():** proporciona una lista con todos los arcos que hay en el grafo. Se puede utilizar con un bucle para recorrer todos los arcos del grafo.

```
cy.edges().forEach(function(ele){
                var target = ele.data('target');
                var source = ele.data('source');
        });
```

- **cy.nodes():** proporciona una lista con todos los nodos que hay en el grafo. Se puede utilizar con un bucle para recorrer todos los nodos del grafo.

```
cy.nodes().forEach(function(ele){
                var id = ele.data('id');
                var name = ele.data('name');
        });
```

- **cy.on():** sirve para ejecutar alguna acción cuando ocurre el evento especificado en la llamada.

```
cy.on('click', 'edge', function(evt){
        alert("Soy un arco");
    });
```

- **cy.data():** se usa para obtener los datos de un elemento.

```
this.data('target');
this.data('source');
```

También se crea un objeto para el menú contextual que según el elemento que se seleccione se mostrará de una manera u otra. Dentro de ese código se pueden especificar las opciones del menú que se quieran tener. A continuación se muestra un ejemplo:

```
cy.cxtmenu({
        selector: 'edge',

        commands: [
            {
                content: '<span class="fa fa-remove fa-2x"></span>',
                select: function(){
                      alert("Opción 1");
                }
            },
                      {
                content: '<span class="fa fa-pencil fa-2x"></span>',
                select: function(){
                    alert("Opción 2");
                }
            }
        ]
    });
cy.cxtmenu({
        selector: 'node',

        commands: [
            {
                content: '<span class="fa fa-flash fa-2x"></span>',
                select: function(){
                    console.log( this.id() );
                }
            },

            {
                content: '<span class="fa fa-star fa-2x"></span>',
                select: function(){
                    console.log( this.data('name') );
                }
            }
        ]
    });
```

En tercer lugar, las principales estructuras de datos utilizadas han sido diccionarios. Se utilizan cuatro diccionarios que se detallan a continuación:

- **infonodes:** en este diccionario se almacenan las descripciones que se recuperan cuando se añade un nodo al grafo. Las claves son los identificadores internos de los nodos que asigna automáticamente la librería Cytoscape.js. El valor de cada clave es el JSON recibido al hacer la llamada de recuperación de la descripción.

- **edgeslabels:** las flechas muestran una etiqueta por defecto que se corresponde con el identicador del nodo del que parten. Este identificador es el que internamente se usa para recuperar del resto de diccionarios la información almacenada. No obstante, se permite reetiquetar una flecha para asignarle un nombre más representativo de la información que proporciona por lo que se necesita almacenar la correspondencia entre la nueva etiqueta y el identificador de nodo al que hace referencia. Esta correspondencia se almacena en este diccionario que toma como claves la nueva etiqueta de una flecha y como valor el identificador de nodo al que se refiere.

- **configurations:** en este diccionario se guardan las configuraciones de los servicios. Las claves son los identificadores internos de los nodos que asigna automáticamente la librería cytoscape.js. El valor de cada clave es una lista en la que se almacenan los valores de la configuración guardada.

- **executions:** en este diccionario se guardan los resultados de ejecución de los servicios. Las claves son los identificadores internos de los nodos que asigna automáticamente la librería Cytoscape.js. El valor de cada clave es el JSON que se recibe como respuesta de la ejecución del servicio.

En cuarto lugar, se ha utilizado la librería JavaScript string-format [24] que sirve para usar los endpoints de los servicios como plantillas en las que se sustituyen automáticamente los valores de los parámetros. Por ejemplo, si tenemos un endpoint como el siguiente:

```
https://zip.getziptastic.com/v2/ES/{postalcode}/{city}
```

y tenemos la siguiente lista de parámetros obtenidos en la configuración:

```
[postalcode, city]
```

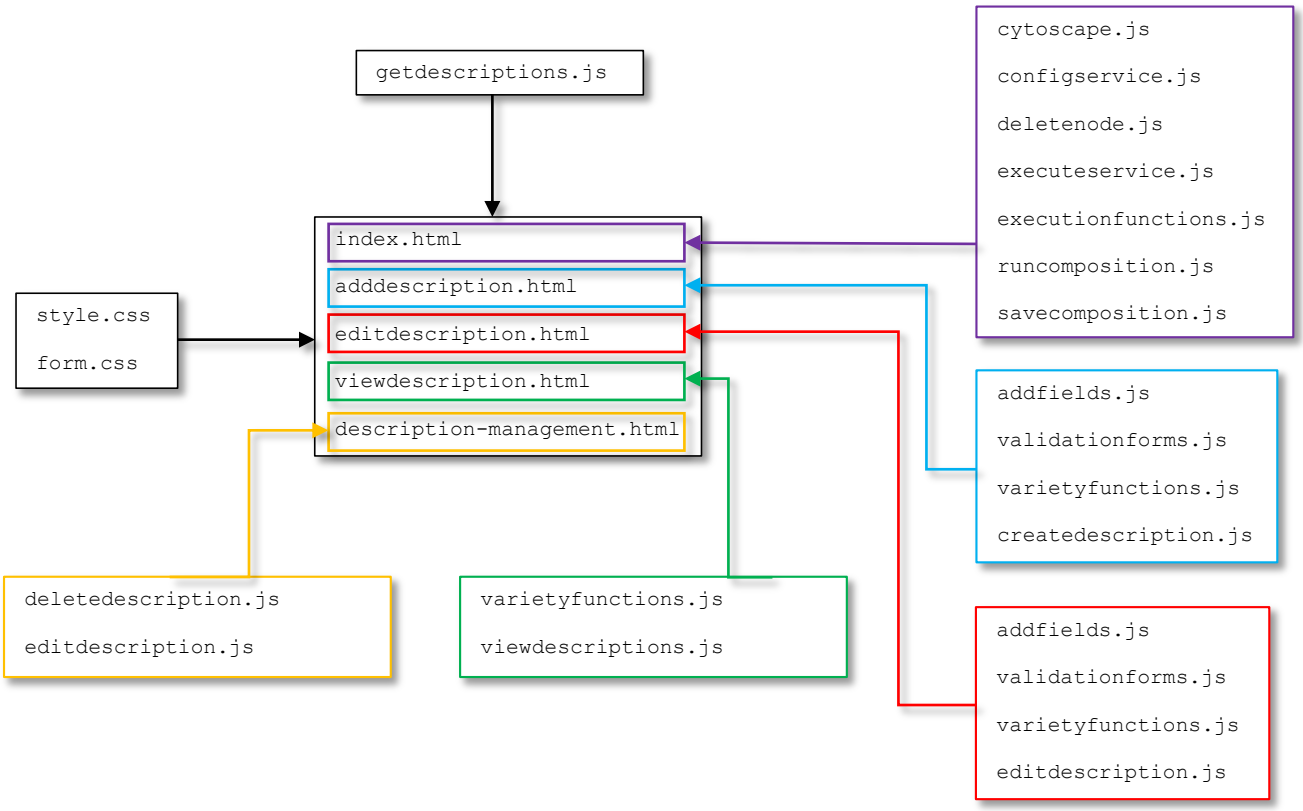
si hacemos:

```
String.format( https://zip.getziptastic.com/v2/ES/{postalcode}/{city}, [postalcode, city])
```

se sustituirán los parámetros cuyo nombre coincida con alguna de las plantillas (lo que está entre llaves). Esto permite automatizar el proceso de creación de los endpoints de llamada con parámetros en el path.

En quinto lugar, cuando se trata de un servicio que requiere autenticación usando una *API Key* (clave de autenticación) ésta se debe insertar directamente en la URI del servicio cuando se describa si ese es el lugar que le corresponde. Si el lugar de la clave es el *path* o la lista de parámetros se podrá poner la clave directamente en el campo correspondiente cuando se configure el servicio. Obviamente, en el proceso de descripción de ese servicio se deberá añadir como parámetro la API Key.

Por último, el código implementado se estructura en tres tipos de ficheros: HTML, JavaScript y CSS. Dentro de las cabeceras de los ficheros HTML se incluyen las referencias a los archivos que se necesiten. A continuación, se puede ver un esquema de la estructura de los ficheros.



En el esquema se ve qué archivos se necesitan en cada uno de los archivos HTML. A parte de los archivos que se muestran también están los correspondientes a las librerías que ya se han comentado anteriormente pero aquí solo se presentan los archivos que se han implementado para el desarrollo de la interfaz. A continuación se describe brevemente cada uno de estos archivos:

- **style.css:** fichero que contiene el código CSS para el diseño de la interfaz.
- **form.css:** archivo que contiene el código CSS para el diseño de los formularios.
- **index.html:** es el archivo principal de la aplicación que contiene el código HTML del módulo de composición.
- **adddescription.html:** fichero que contiene el código HTML del formulario para la creación de una descripción de un servicio.
- **editdescription.html:** fichero que contiene el código HTML del formulario para la modificación de una descripción de un servicio.
- **viewdescription.html:** archivo que contiene el código HTML para mostrar la tabla con la información de una descripción de un servicio.
- **description-management.html:** archivo que contiene el código HTML del módulo de gestión de descripciones.
- **getdescriptions.js:** en este fichero se encuentran las funciones para recuperar las listas de servicios simples y compuestos.
- **deletedescription.js:** en este archivo se encuentran las funciones para eliminar la descripción de un servicio simple y de un servicio compuesto.
- **editdescription.js:** este fichero contiene las funciones para recuperar la información de una descripción y modificarla.
- **varietyfunctions.js:** este archivo contiene funciones variadas de uso general que no tienen que ver directamente con la funcionalidad de la aplicación.
- **viewdescription.js:** en este fichero se encuentra la función para recuperar la información de una descripción y visualizarla.
- **addfields.js:** en este archivo se encuentra la función necesaria para permitir añadir parámetros de forma dinámica en los formularios de creación y modificación de una descripción.
- **validationforms.js:** este fichero contiene la función para validar los formularios de creación y modificación de una descripción.
- **createdescription.js:** este archivo contiene la función para crear una descripción de un servicio.
- **cytoscape.js:** en este archivo se encuentran todas las funciones relacionadas con el uso del grafo.
- **configservice.js:** en este fichero se encuentra la función para la configuración de un servicio.
- **deletenode.js:** este archivo contiene la función para eliminar un nodo del grafo.
- **executeservice.js:** este fichero contiene la función para saber si hay que iterar un grafo y controlar así las veces que se tiene que llamar a la función de ejecución.

- **executionfunctions.js:** en este archivo se encuentra la función que se encarga de realizar la llamada correspondiente para ejecutar un servicio.

- **runcomposition.js:** en este archivo se encuentra la función que se encarga de ejecutar un servicio compuesto.
- **savecomposition.js:** este fichero contiene la función que guarda el grafo de un servicio compuesto.

Esta aplicación no requiere ningún tipo de instalación para ejecutar los archivos. Simplemente hay que copiar la carpeta que contiene todos los archivos del proyecto y abrir el archivo index.html con un navegador para iniciar la interfaz.

3.4 Pruebas

Después de ver cómo se ha diseñado e implementado la interfaz, se especifican a continuación las pruebas que se han llevado a cabo para comprobar el correcto funcionamiento de los módulos de la aplicación.

En el módulo de gestión de descripciones de servicios se han realizado las siguientes pruebas:

- Se han añadido descripciones y se ha comprobado que se crean correctamente. El formulario obliga a introducir los campos obligatorios "name" y "endpoint", por lo que no se puede enviar si no se introducen esos datos. Además si el nombre del servicio que se introduce ya existe se avisa de ello con un mensaje.

- Se han eliminado descripciones y se ha comprobado que se eliminan correctamente. En el caso de que haya algún problema se muestra el mensaje que devuelve el servidor (podría ocurrir que se estropeara la base de datos del servidor y no se encontrara la descripción, entonces el servidor diría que no se encuentra).

- Se han modificado algunas descripciones y se ha comprobado que se modifican correctamente. Como ocurre en la creación, los campos de "name" y "endpoint" son obligatorios. Además si se modifica el nombre del servicio y este ya existe se avisa de ello con un mensaje.

- Se han visualizado descripciones y se ha comprobado que se recupera correctamente su información y que se visualizan adecuadamente.

En el módulo de composición se han realizado las siguientes pruebas:

- Se ha comprobado que los servicios cuya descripción se ha añadido en el módulo de gestión de descripciones aparecen en las listas de servicios del módulo de composición. Si no existe ninguna descripción se avisa de ello.

- Se ha añadido un servicio al área de composición arrastrando y soltando el nombre del servicio de la lista de servicios y se ha comprobado que se crea correctamente el nodo que representa al servicio.

- Se ha comprobado que cuando se añade un servicio al área de composición y la llamada HTTP GET para obtener su información falla se avisa del error con un mensaje y se borra el nodo del grafo.

- Se han etiquetado flechas y se ha visto que el nombre de la etiqueta cambia adecuadamente.

- Se ha probado a etiquetar una flecha con el mismo nombre de otra etiqueta o con la etiqueta en blanco y se ha visto que la interfaz no lo permite y que avisa de ello.

- Se han eliminado elementos del grafo mediante la opción correspondiente del menú contextual y se ha comprobado que se borran correctamente.

- Se ha eliminado un grafo mediante el botón "Delete graph" y se ha visto que se el grafo borra correctamente.

- Se ha pulsado en botón "Delete graph" cuando no había ningún grafo dibujado y se ha comprobado que la interfaz avisa de que está vacío.

- Se ha visualizado la información de un servicio mediante la opción correspondiente del menú contextual y se ha visto que se muestra correctamente.

- Se ha configurado un servicio y se ha comprobado que la configuración se almacena y se recupera correctamente.

- Se ha intentado ejecutar un servicio con parámetros que no había sido configurado y se ha visto que la interfaz no lo ha permitido y que ha avisado de ello con un mensaje.

- Se ha intentado ejecutar un servicio que tenía nodos predecesores conectados sin ejecutar y se ha visto que la interfaz no lo permite y que avisa de ello con un mensaje.

- Se ha intentado ejecutar un servicio en cuya configuración se hacía referencia a los resultados de un nodo que no está conectado a él y se ha visto que la interfaz no lo permite y que avisa de ello con un mensaje.

- Se ha ejecutado un servicio que está configurado correctamente y se ha comprobado que se ejecuta adecuadamente.

- Se ha ejecutado un servicio compuesto mediante el botón "Run composition" y se ha visto que se ejecuta correctamente.

- Se ha intentado ejecutar una composición vacía mediante el botón "Run composition" y se ha visto que la interfaz no lo permite y avisa de ello con un mensaje.

- Se han visualizado los resultados de ejecución de un servicio mediante la opción correspondiente del menú contextual y se ha comprobado que se visualizan correctamente.

- Se ha ejecutado un servicio configurado correctamente para iterar y se ha visto que se ejecuta las veces que corresponden y que se almacenan todos los resultados.

- Se ha intentado ejecutar un servicio configurado para iterar y que hace referencia a unos resultados que no son ni un JSON ni una lista y se ha visto que la interfaz no lo permite y que avisa de ello con un mensaje.

- Se ha guardado un servicio compuesto y se ha comprobado que se almacena correctamente y que aparece listado en la lista de servicios compuestos del módulo de composición y en la lista de descripciones de servicios compuestos del módulo de gestión de descripciones.

- Se ha intentado guardar una composición vacía y se ha visto que la interfaz no lo permite y que avisa de ello con un mensaje.

- Se ha recuperado un servicio compuesto de la lista de servicios compuestos arrastrándolo al área de composición y se ha comprobado que se recupera correctamente y que se dibuja el grafo del servicio.

- Se ha intentado recuperar un servicio compuesto con alguno de los servicios que lo componen eliminado de la lista de servicios simples y se ha comprobado que ese nodo aparece en color rojo y que la interfaz

avisa del error con un mensaje. Además, también se ha visto que si alguno de los servicios simples que componen el servicio compuesto falla al hacer la llamada para obtener su información la interfaz también lo notifica y lo colorea en rojo.

3.5 Escenario de uso

Como escenario de uso se va a imaginar que se quiere obtener una lista de libros relacionados con el lenguaje de programación JavaScript y que solo se quieren obtener libros que tengan buenas críticas. Se va a imaginar también que en el repositorio de servicios web no existe ningún servicio que haga esto. Sin embargo, si que existe una serie de servicios que combinados permiten obtener un servicio compuesto que si ofrece el resultado deseado.

En la siguiente tabla se especifican los servicios simples que se pueden utilizar para componer el nuevo servicio:

Servicio	Parámetros de entrada	Parámetros de salida	Descripción
Search IT Books	*Query:* palabra clave. *Page:* número de páginas de resultados que se quieren ver.	Un JSON que entre otras clave contiene una llamada *Books* que es una lista de los libros encontrados.	Servicio que a partir de una palabra clave busca libros relacionados con esa palabra dentro del área de las TI.
Get Criticals	*isbn:* el número ISBN de un libro.	Un JSON que contiene el *rating* de la crítica junto con el *isbn* del libro al que corresponde esa crítica.	Servicio que a partir del ISBN de un libro busca sus críticas y devuelve el rating correspondiente.
FilterCriticals	*criticalbooklist:* lista de libros. *minrating:* rating mínimo que tiene que tener una crítica.	Una lista en la que cada elemento es un JSON que contiene el ISBN de un libro y el rating de ese libro.	Servicio que a partir de una lista de libros y un rating de crítica filtra solo aquellos libros que tienen un rating igual o superior al indicado.
Google Books	*isbn:* el número ISBN de un libro.	Un JSON que contiene toda la información sobre el libro indicado.	Servicio que a partir de un ISBN te devuelve toda la información que tiene ese libro (autores, título, etc).
Filter Results	*googlebook:* JSON con la información que proporciona Google Books sobre un libro.	Un JSON que contiene las características más importantes de un libro.	Servicio que a partir de la información proporcionada por Google Book sobre un libro devuelve un JSON con lo más importante de ese libro.

Tabla 3: Servicios utilizados en el escenario de ejemplo

Para obtener el resultado deseado, estos servicios simples que se acaban de especificar se pueden ordenar en un grafo de composición como el que se muestra en la figura 16.

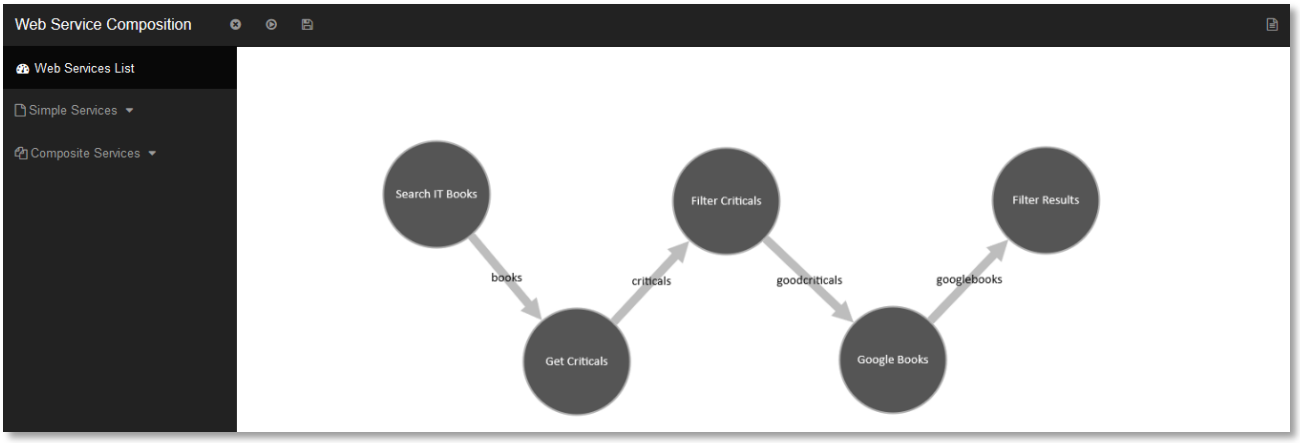


Figura 16: Grafo del escenario de uso

Tal y como se ve en la figura, en primer lugar, se ejecuta *Search IT Books* que obtiene una lista de libros que coinciden con la búsqueda. Después, se ejecuta *Get Criticals* que a partir del isbn de cada libro de la lista de resultados obtiene el rating de la crítica. Seguidamente, se ejecuta *Filter Criticals* que a partir de la lista de libros con rating devuelve otra lista que contiene únicamente los libros que tienen un rating de crítica superior al indicado. A continuación, se ejecuta *Google Books*, que usa el número ISBN de la lista de críticas para buscar el libro. Finalmente, se ejecuta *Filter Results* que hace un resumen de las características más importantes de un libro.

A continuación se explica cómo se ha configurado cada servicio y se muestran los resultados de ejecución de cada uno de ellos.

Servicio Search IT Books

Para configurar este servicio (ver figura 17) se introduce en el parámetro *query* la palabra clave "javascript" ya que queremos buscar libros sobre ese tema. En el parámetro *number* introducimos el valor "1" para indicar que solo nos interesa ver los resultados de la primera página. Como se trata de datos que introducimos de forma manual se selecciona la opción "Manual input".

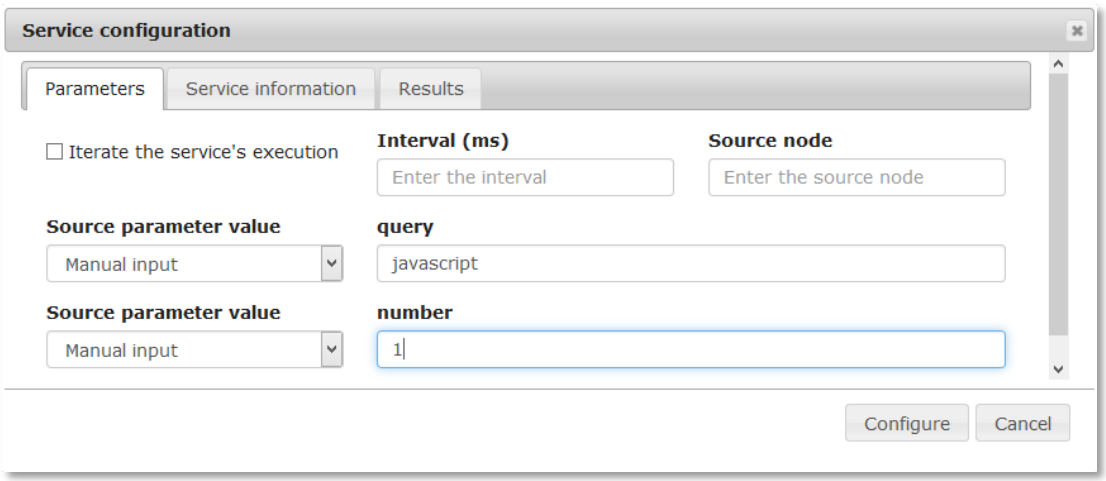


Figura 17: Configuración del servicio Search IT Books

Después de configurar el servicio, se ejecuta y se obtiene como resultado (ver figura 18) un JSON del que nos interesa en particular la clave *Books*, que contiene una lista con los libros relacionados con la temática introducida.

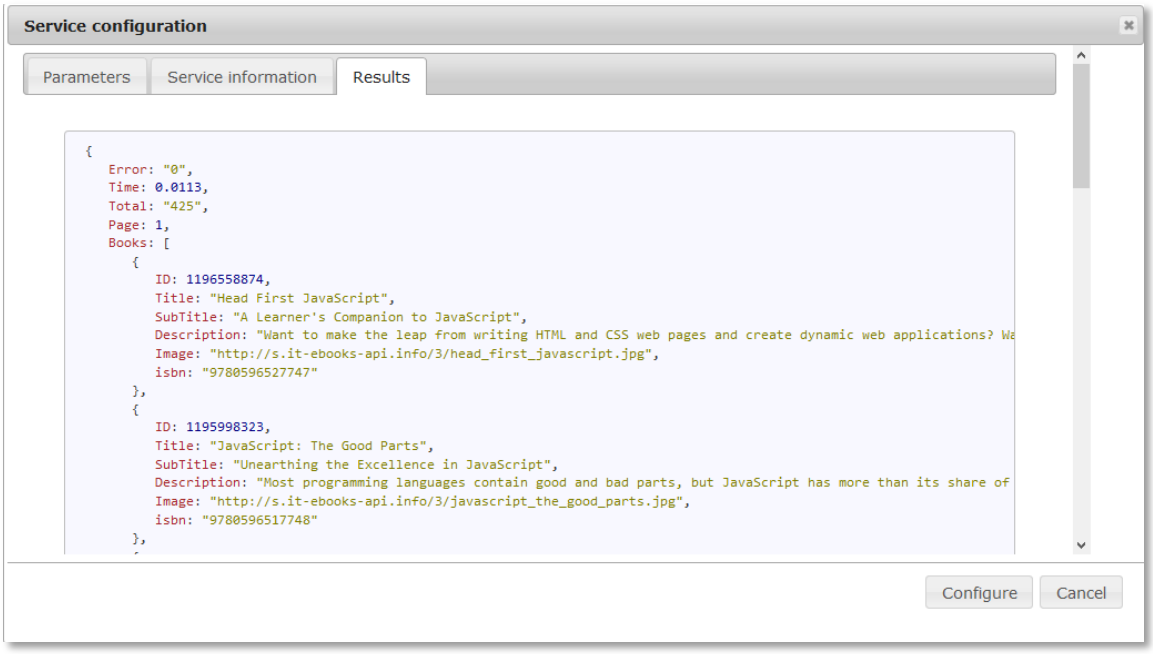


Figura 18: Resultados de ejecución del servicio Search IT Books

Get Criticals

A partir de los resultados de ejecución de *Search IT Books* se obtiene una lista de libros relacionados con el lenguaje de programación Javascript. Ahora, con el servicio *Get Criticals* lo que se quiere hacer es obtener una crítica por cada libro de la lista. Para ello hace falta introducir el número ISBN de cada libro, por lo que el servicio *Get Criticals* deberá ejecutarse una vez por cada libro de la lista. Como este servicio se tiene que ejecutar una vez por cada libro de la lista se tiene que marcar la casilla de que es iterable y se debe indicar un tiempo de espera entre llamadas y la fuente sobre la que se va a iterar. Como se ve en la figura 16 los resultados de *Search IT Books* se representan mediante la etiqueta de la flecha que conecta estos dos nodos y que en este caso es *Books*. En la figura 19 se puede ver la expresión JSON que hay que poner tanto en el parámetro *Source node* como en el parámetro *isbn* para acceder a la parte del JSON que interesa utilizar como valor de entrada.

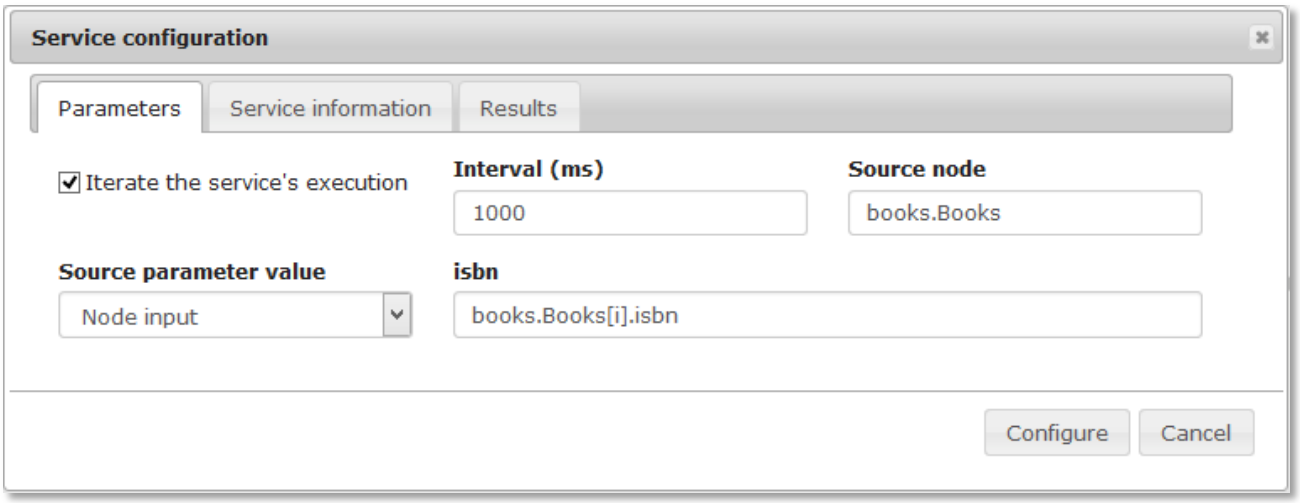


Figura 19: Configuración del servicio Get Criticals

Después de configurar el servicio, se ejecuta y se obtiene como resultado (ver figura 20) una lista de JSONs donde cada JSON es una crítica que contiene el número ISBN del libro y su rating correspondiente.

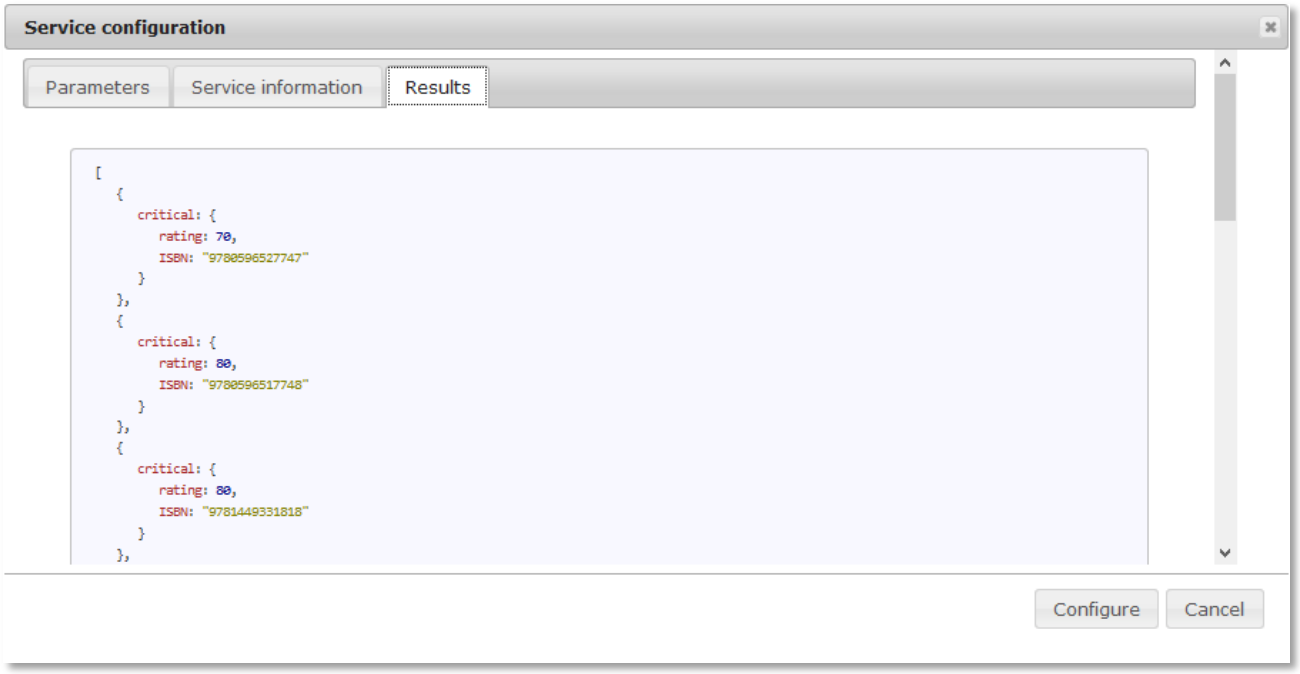


Figura 20: Resultados de la ejecución del servicio Get Criticals

Filter Criticals

Una vez obtenidas las críticas de los libros ahora se quieren filtrar para quedarse solo con aquellos libros que superen un rating mínimo de calidad literaría. Este servicio toma como entrada la lista de críticas obtenida del nodo anterior y devuelve otra lista en la que solo están los libros que superan el rating mínimo. Para configurar este servicio (ver figura 21) se introduce en el campo *rating* un número, que en este caso será por ejemplo 70 (este valor es manual). Después en el campo *criticalbooklist* se introduce la expresión JSON necesaria para obtener la lista de críticas de los resultados de ejecución de *Get Criticals*. Se puede acceder a los resultados de *Get Criticals* con la etiqueta *Criticals* (ver figura 16).

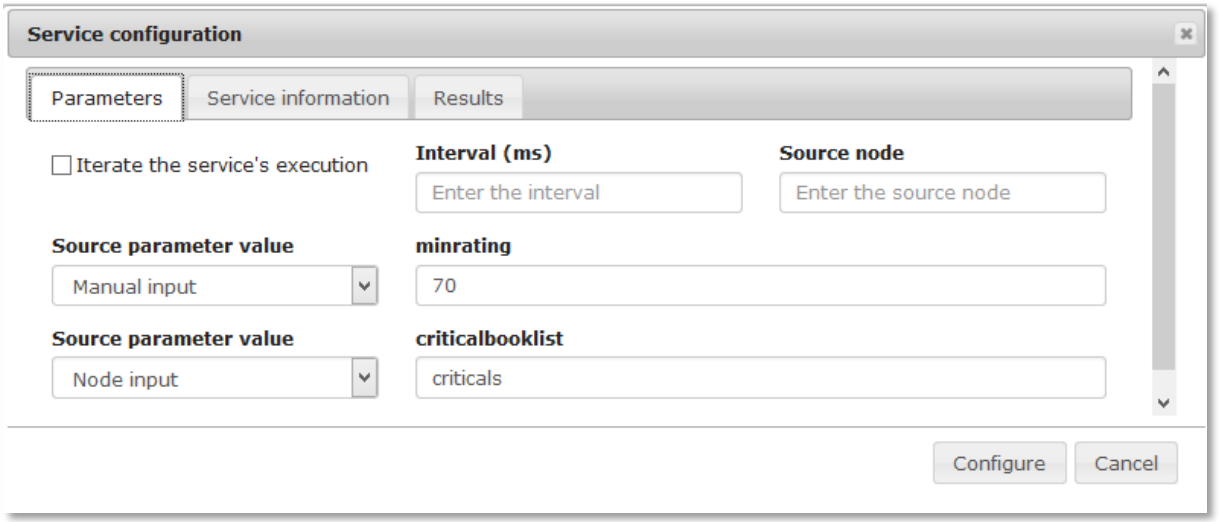


Figura 21: Configuración del servicio Filter Criticals

Después de ejecutarlo se obtiene como resultado (ver figura 22) una lista de JSONs donde cada JSON es una crítica que contiene el número ISBN del libro y su rating correspondiente pero con la diferencia de que solo están los libros que tienen un rating igual o superior al rating mínimo (en este caso 70).

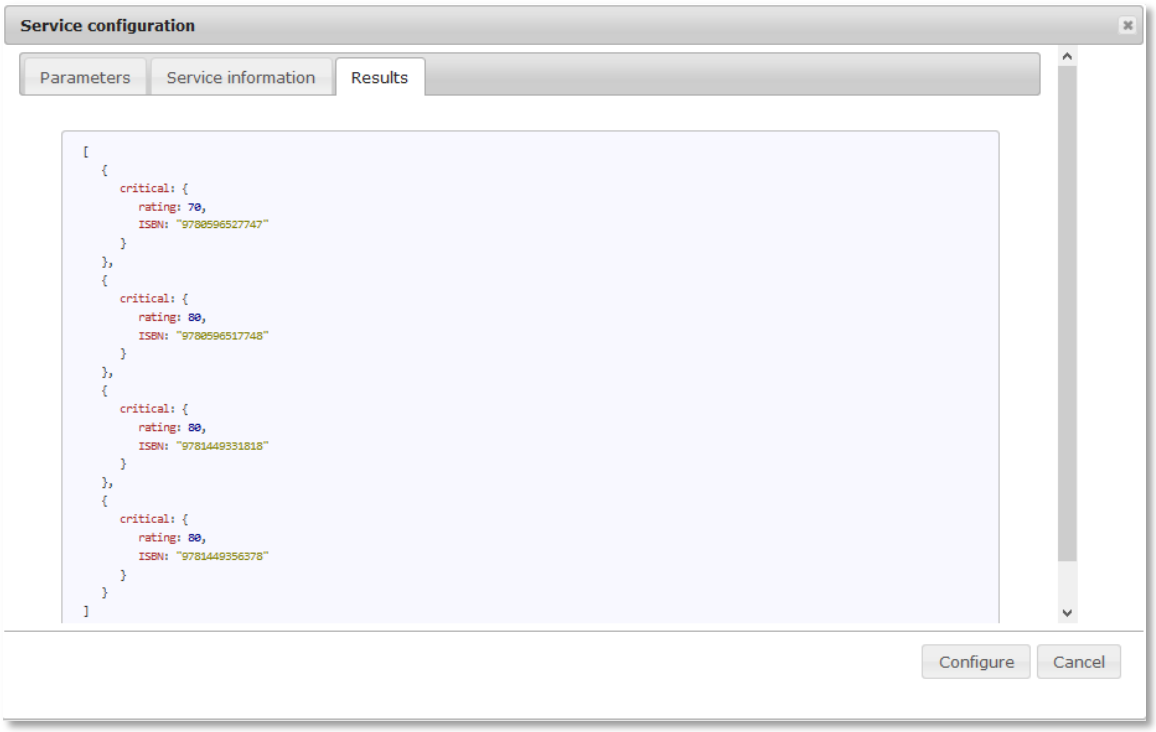


Figura 22: Resultados de ejecución del servicio Filter Results

Google Books

Ahora que se han obtenido los libros relacionados con JavaScript que tienen un rating mínimo de 70, el siguiente paso es buscarlos. Para esto se utiliza el servicio *Google Books* que partiendo del ISBN de un libro obtiene información sobre el. Este servicio se tiene que ejecutar una vez por cada libro así que hay que marcarlo para que itere. También hay que indicar el tiempo de espera entre iteraciones y la fuente sobre la que se va a iterar. En este caso se puede acceder a los resultados del nodo anterior mediante la etiqueta *goodcriticals* (ver figura 16). La configuración de este servicio se puede ver en la figura 23.

Figura 23: Configuración del servicio Google Books

Los resultados de ejecución se muestran en la figura 24. Como se puede ver se obtiene una lista de JSONs que contiene mucha información sobre cada uno de los libros. En concreto, la clave *ítems* contiene una lista de libros que coinciden con el libro buscado. Como normalmente la primera coincidencia suele ser la acertada en el servicio *Filter Results* se mostrará solo esa.

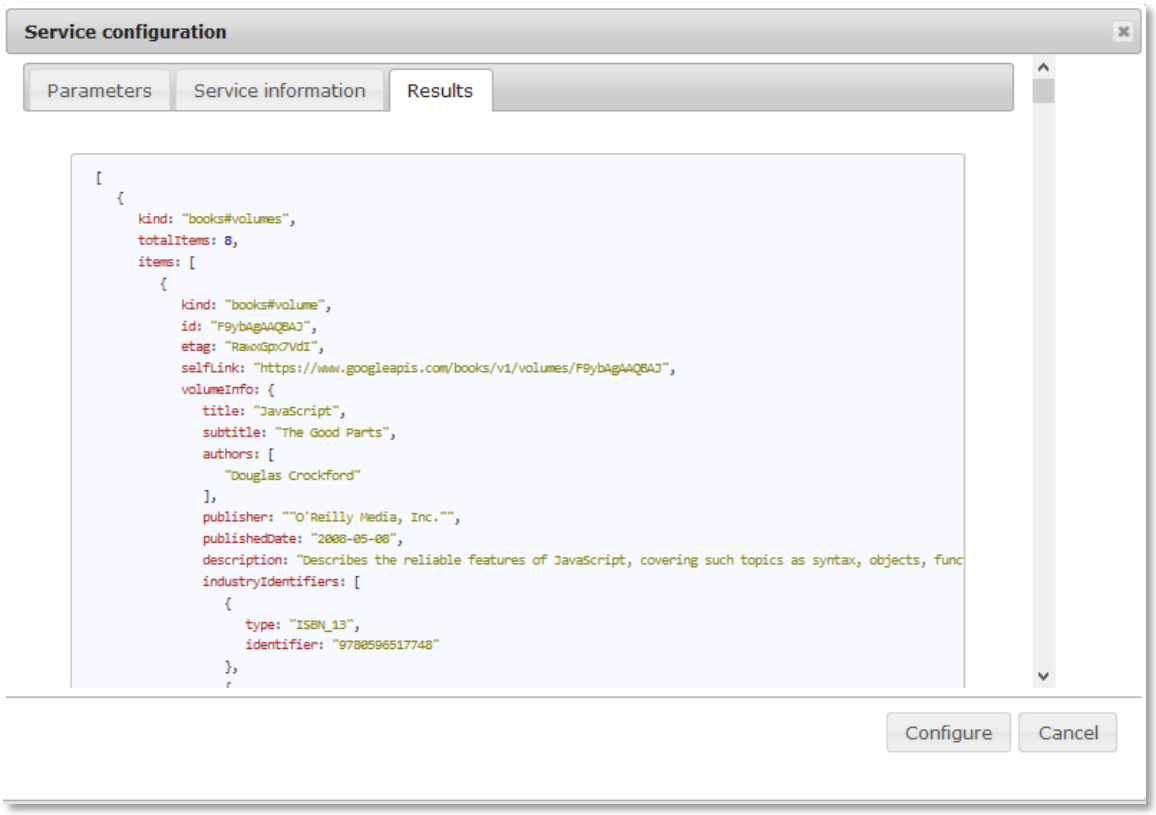


Figura 24: Resultados de ejecución del servicio Google Books

Filter Results

Finalmente, se ejecuta este servicio. Como los resultados que se obtienen por cada libro son demasiado extensos se utiliza este servicio que selecciona simplemente los datos más relevantes. Toma como valor de entrada la lista de ítems de cada libro y devuelve como resultado un JSON que contiene los datos principales de cada libro: autores, título, descripción y url donde se puede encontrar el libro. Este servicio se ejecutará una vez por cada libro de la lista por lo que habrá que marcarlo como iterable, indicar el tiempo de espera y la fuente sobre la que se iterará. En este caso se puede acceder a los resultados del nodo anterior mediante la etiqueta *googlebooks* (ver figura 16). La configuración de este servicio se muestra en la figura 25.

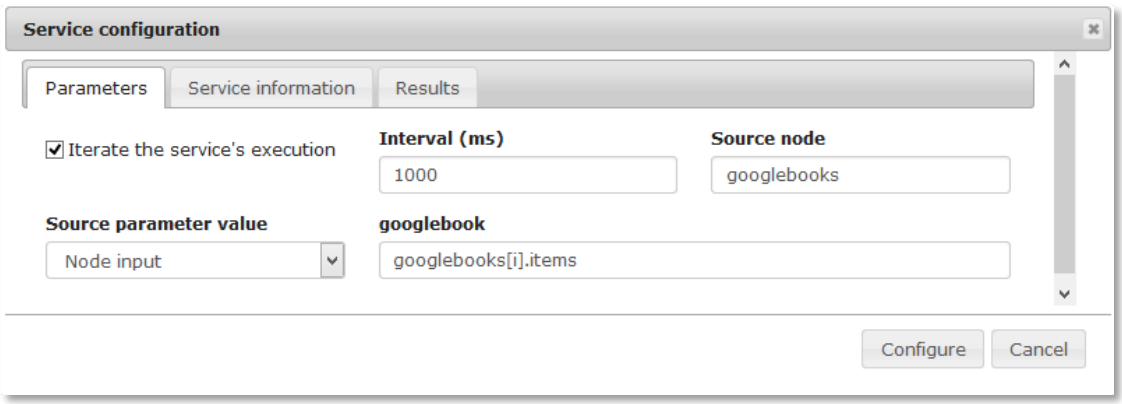


Figura 25: Configuración del servicio Filter Results

Los resultados de ejecución se muestran en la figura 26. Como se puede ver se obtiene una lista de JSONs donde cada uno de ellos contiene el título, los autores, la descripción y la url de un libro.

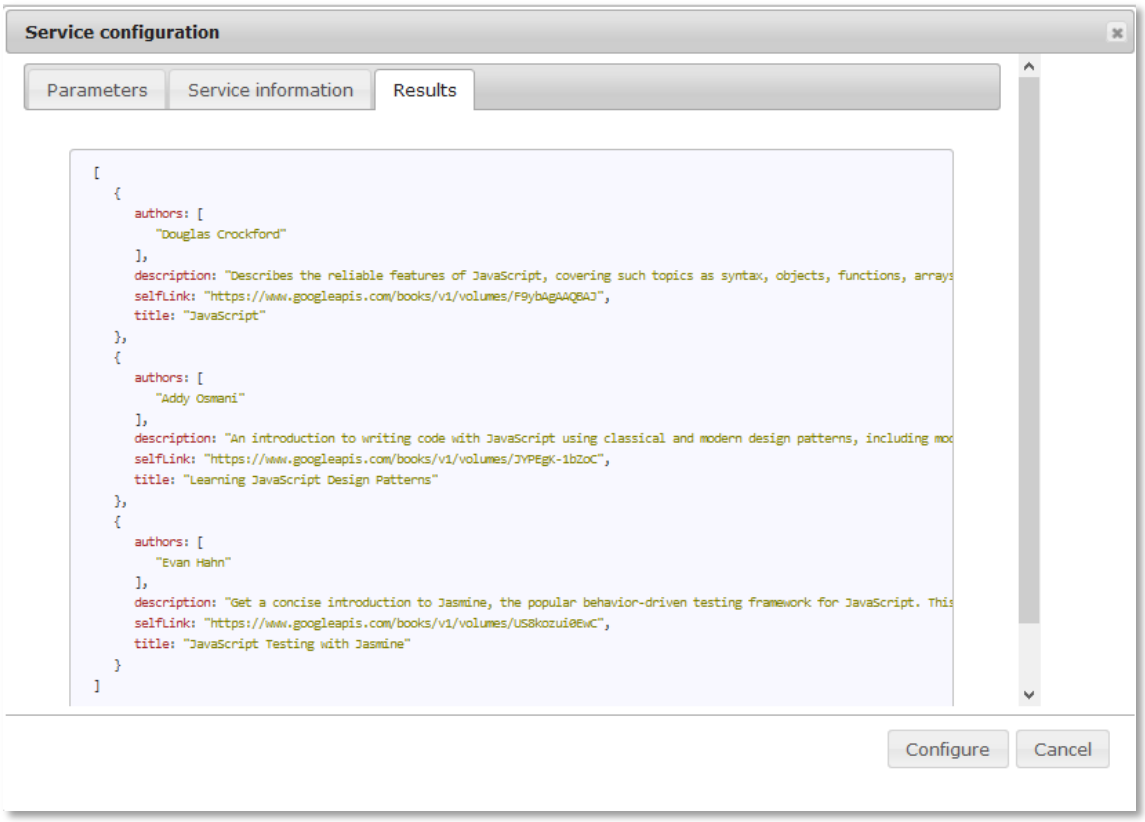


Figura 26: Resultados de ejecución del servicio Filter Results

4 CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se presentan las conclusiones del proyecto así como el trabajo futuro que se podría realizar para mejorar la interfaz desarrollada.

4.1 Conclusiones

En este trabajo de fin de máster se ha presentado el desarrollo de una interfaz web para la composición manual de servicios web basados en el estilo arquitectónico REST que cumple con los objetivos establecidos.

En el estado del arte, se ha visto que REST es un estilo arquitectónico ligero y simple que nos permite aprovechar las características de la Web que la han hecho tan exitosa. Debido a las ventajas que ofrece, como una mayor escalabilidad o el uso de interfaces uniformes, ha aumentado su uso en detrimento del protocolo SOAP, más pesado y complejo. Esto ha propiciado el aumento de los servicios REST, por lo que se necesitan mecanismos que permitan reutilizar los servicios existentes usando algún proceso de composición. Dado que el proceso de composición automático es muy complejo y la mayoría de enfoques proponen soluciones manuales o semiautomáticas, en este proyecto se utiliza composición manual basada en la técnica de flujos de trabajo (workflow-based) que permite la ejecución secuencial de los servicios.

Finalmente, durante el desarrollo de la interfaz, se ha visto que las descripciones de servicios permiten saber qué hace un servicio, cómo se ejecuta, qué parámetros tiene o dónde se encuentra. Esto facilita usar los servicios en un proceso de composición como el utilizado en este proyecto. Además, también se ha visto que la composición de servicios web permite reutilizar servicios para crear uno nuevo que resuelva un problema que los servicios existentes no resuelven por si solos.

4.2 Trabajo futuro

La composición de servicios web RESTful es un campo reciente que todavía no ha sido ampliamente estudiado y por lo tanto no existen muchos enfoques. Sin embargo, actualmente si que existen muchos enfoques que abordan el problema de la composición de servicios web semánticos (SWS). La investigación sobre los SWS se ha dedicado a tratar de reducir el esfuerzo manual necesario para la manipulación de los servicios web. Se pretende que la tarea de la composición de servicios web tenga un mayor nivel de automatización gracias a las descripciones semánticas de las propiedades de los servicios web. Sin embargo, la mayoría de las descripciones de los servicios RESTful existentes no tienen anotaciones semánticas que les permitan ser legibles por máquina y por tanto automatizables. Teniendo en cuenta esto, el trabajo futuro de este proyecto consistiría por un lado, en permitir la descripción semántica de los servicios, y por otro lado usar esta semántica para buscar servicios y automatizar el proceso de composición utilizando alguna de las técnicas que existen.

Además, se podrían hacer algunas mejoras en la interfaz como elegir los datos de entrada y de salida de un nodo sin tener que escribir una expresión JSON, exportar los resultados de ejecución para visualizarlos de forma más cómoda y añadir un buscador para encontrar servicios.

Referencias

[1] https://es.wikipedia.org/wiki/Representational_State_Transfer

[2] https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol

[3] http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

[4] http://bibing.us.es/proyectos/abreproy/11247/fichero/Memoria%252F8-Representational+State+Transfer+%28REST%29.pdf

[5] Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

[6] http://es.wikipedia.org/wiki/Servicios_Web

[7] http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

[8] Pautasso, C. (2014). RESTful web services: principles, patterns, emerging technologies. In *Web Services Foundations* (pp. 31-51). Springer New York.

[9] Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. Information Sciences, 280, 218-238.

[10] Kona, S., Bansal, A., Gupta, G., & Hite, D. (2007, July). Automatic Composition of SemanticWeb Services. In *ICWS* (Vol. 7, pp. 150-158).

[11] Rostami, N. H., Kheirkhah, E., & Jalali, M. (2013). WEB SERVICES COMPOSITION METHODS AND TECHNIQUES: A REVIEW. International Journal of Computer Science, Engineering & Information Technology, 3(6).

[12] Thai, M. T., & Pardalos, P. M. (2012). Handbook of optimization in complex networks. Springer.

[13] Zain, Jasni Mohamad, Wan Mohd, Wan Maseri & El-Qawasmeh (2011). Software Engineering and Computer Systems, Part III. Springer.

[14] Li, Z., & O'Brien, L. (2011). Towards effort estimation for web service compositions using classification matrix. International Journal On Advances in Internet Technology, 3(3 and 4), 245-260.

[15] Baryannis, G., & Plexousakis, D. (2010). Automated Web Service Composition: State of the Art and Research Challenges. ICS-FORTH, Tech. Rep, 409.

[16] Pessini, E. C. Expressiveness of Automatic Semantic Web Service Composition Approaches: A Survey based on Workflow Patterns. Revista de Informática Teórica e Aplicada, 21(1), 45-76.

[17] Sara Valero Menacho (2014). Descubrimiento y composición semántica de servicios web REST. Universitat Jaume I.

[18] Bennara, M., Mrissa, M., & Amghar, Y. (2014, April). An approach for composing RESTful linked services on the web. In Proceedings of the companion publication of the 23rd international conference on World wide web companion (pp. 977-982). International World Wide Web Conferences Steering Committee.

[19] Alarcon, R., Wilde, E., & Bellido, J. (2011). Hypermedia-driven RESTful service composition. In Service-Oriented Computing (pp. 111-120). Springer Berlin Heidelberg.

[20] Zhao, H., & Doshi, P. (2009, July). Towards automated restful web service composition. In Web Services, 2009. ICWS 2009. IEEE International Conference on (pp. 189-196). IEEE.

[21] Pautasso, C. (2009). RESTful Web service composition with BPEL for REST. Data & Knowledge Engineering, 68(9), 851-866.

[22] Rosenberg, F., Curbera, F., Duftler, M. J., & Khalaf, R. (2008). Composing restful services and collaborative workflows: A lightweight approach. Internet Computing, IEEE, 12(5), 24-31.

[23] Pautasso, C. (2009, January). Composing restful services with jopera. In Software Composition (pp. 142-159). Springer Berlin Heidelberg.

[24] https://closure-library.googlecode.com/git-history/docs/local_closure_goog_string_stringformat.js.source.html

[25] http://js.cytoscape.org

[26] http://sswap.info

[27] https://www.mashape.com

Anexo I: manual de usuario

En este anexo se presenta el manual de usuario de la interfaz web. Se divide en dos partes, una para el módulo de gestión de descripciones y otra para el módulo de composición de servicios.

Manual de uso del módulo de gestión de descripciones

El módulo de gestión de descripciones se utiliza para gestionar las descripciones de los servicios que se quieren utilizar en el módulo de composición. En este módulo se pueden ver listados de las descripciones de los servicios simples y compuestos que se han dado de alta en el sistema. También se pueden realizar tareas tales como crear, modificar, eliminar y visualizar una descripción. A partir de la pantalla principal (ver figura 27) se puede acceder a todas estas funcionalidades. A continuación se detalla como realizar cada una de estas tareas.

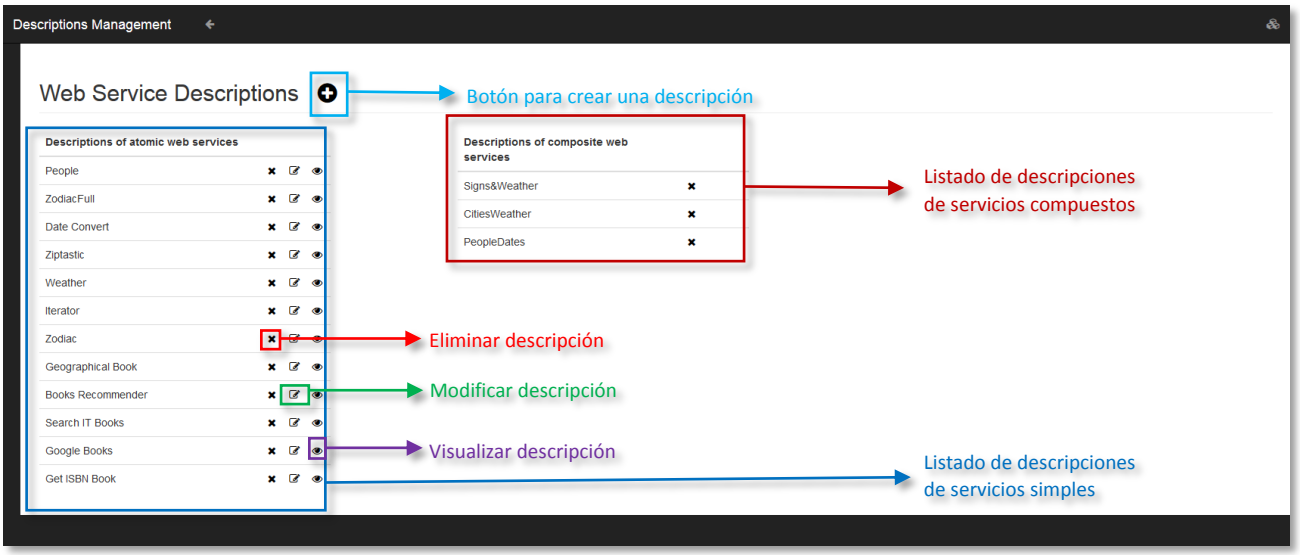


Figura 27: Pantalla principal módulo de gestión de descripciones

Crear una descripción

Para crear una nueva descripción hay que pulsar en el botón de crear una descripción (ver figura 27). Al pulsar en este botón aparecerá un formulario (ver figura 28) en el que se deberá rellenar como mínimo el nombre del servicio que se está describiendo y su endpoint. Si no se rellenan estos campos obligatorios la interfaz no permitirá enviar el formulario. Si queremos añadir más parámetros solo hay que pulsar en el botón de añadir parámetros (ver figura 28). En la tabla 3 se puede ver el significado de cada uno de los campos del formulario. Una vez completado el formulario deberemos pulsar en el botón de envío "Create description" (ver figura 28) y si todo ha ido bien se recibirá un mensaje de confirmación del servidor. En caso contrario se recibirá un mensaje indicando el posible error.

Modificar una descripción

Para modificar una descripción hay que pulsar sobre el botón de modificar descripción (ver figura 27). Al pulsar en este botón aparecerá el mismo formulario que en el caso de la creación (ver figura 28) con los mismos campos (ver tabla 4) pero con la diferencia que en este formulario aparecerán en los campos los valores que estuvieran almacenados en la descripción. A continuación se podrán hacer las modificaciones pertinentes y una vez terminada la modificación se deberá pulsar en el botón "Modify description". Si todo ha ido bien se recibirá un mensaje de confirmación del servidor, en caso contrario se recibirá un mensaje indicando el posible error.

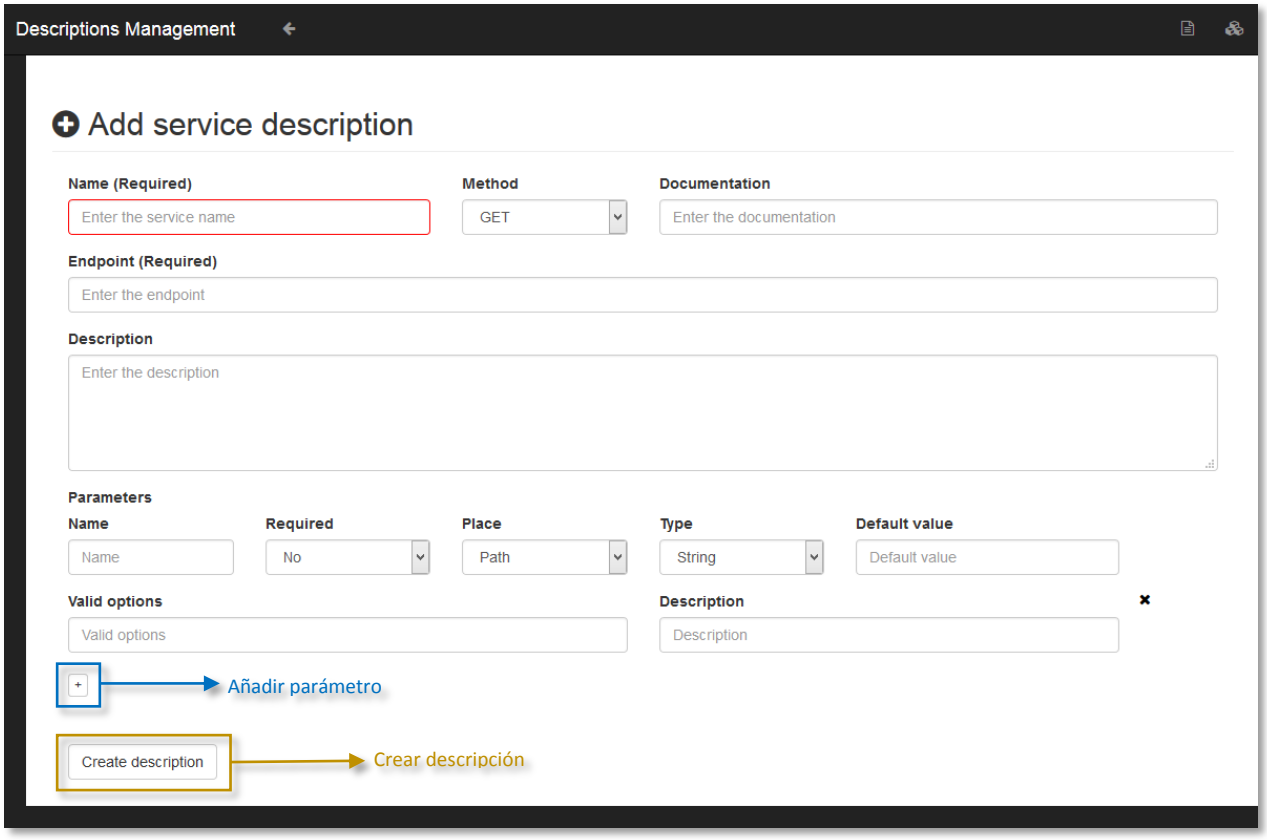


Figura 28: Formulario de creación de una descripción

Campo	Significado	Valores
Name	Nombre del servicio que se quiere describir. No tiene porque coincidir con el nombre real pudiéndose escribir un nombre que nos resulte más representativo del servicio. Es un campo obligatorio ya que todos lo servicios deben tener un nombre que los represente cuando se cree el grafo de composición.	
Method	Indica el método HTTP que se utiliza para llamar al servicio.	GET, POST, PUT, DELETE
Documentation	Es la url del servicio en la que se encuentra su documentación.	
Endpoint	Es la url del servicio que lo identifica como recurso. Se usa para acceder al recurso mediante la correspondiente llamada HTTP. Este campo es obligatorio pues sin endpoint no se puede acceder al servicio.	
Description	Descripción de lo que hace el servicio.	
Campos de los parámetros		
Name	Nombre del parámetro. Debe coincidir con el nombre real del parámetro correspondiente del servicio.	
Required	Indica si el parámetro es obligatorio o no.	Yes, No
Place	Indica el lugar en el que se envía el parámetro.	Path, parameter, payload
Type	Indica el tipo de dato del parámetro.	String, int, float, date, other
Default value	Indica el posible valor por defecto que puede tomar el parámetro si no se rellena.	
Valid options	Indica lasposibles opciones que acepta el parámetro.	
Description	Descripción de lo que significa el parámetro.	

Tabla 4: Campos del formulario de creación y modificación de una descripción

Eliminar una descripción

Para eliminar una descripción hay que pulsar sobre el botón de eliminar descripción (ver figura 27). Aparecerá un cuadro de diálogo (ver figura 29) pidiendo confirmación. Si se pulsa sobre "Aceptar" la descripción se eliminará, y si se pulsa sobre "Cancelar" no se efectuará la eliminación. Si todo ha ido bien se recibirá un mensaje de confirmación del servidor, en caso contrario se recibirá un mensaje indicando el posible error.

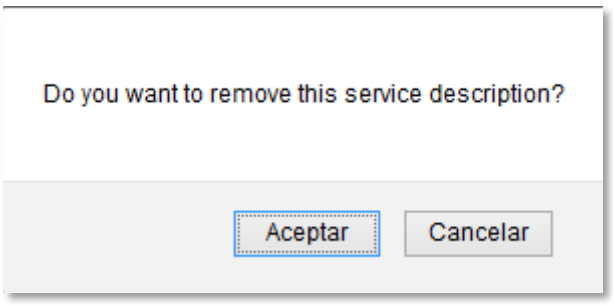


Figura 29: Diálogo de confirmación de eliminación de una descripción

Visualizar una descripción

Para visualizar una descripción hay que pulsar sobre el botón de visualizar descripción (ver figura 27). Aparecerá una tabla (ver figura 30) con toda la información almacenada en la descripción.

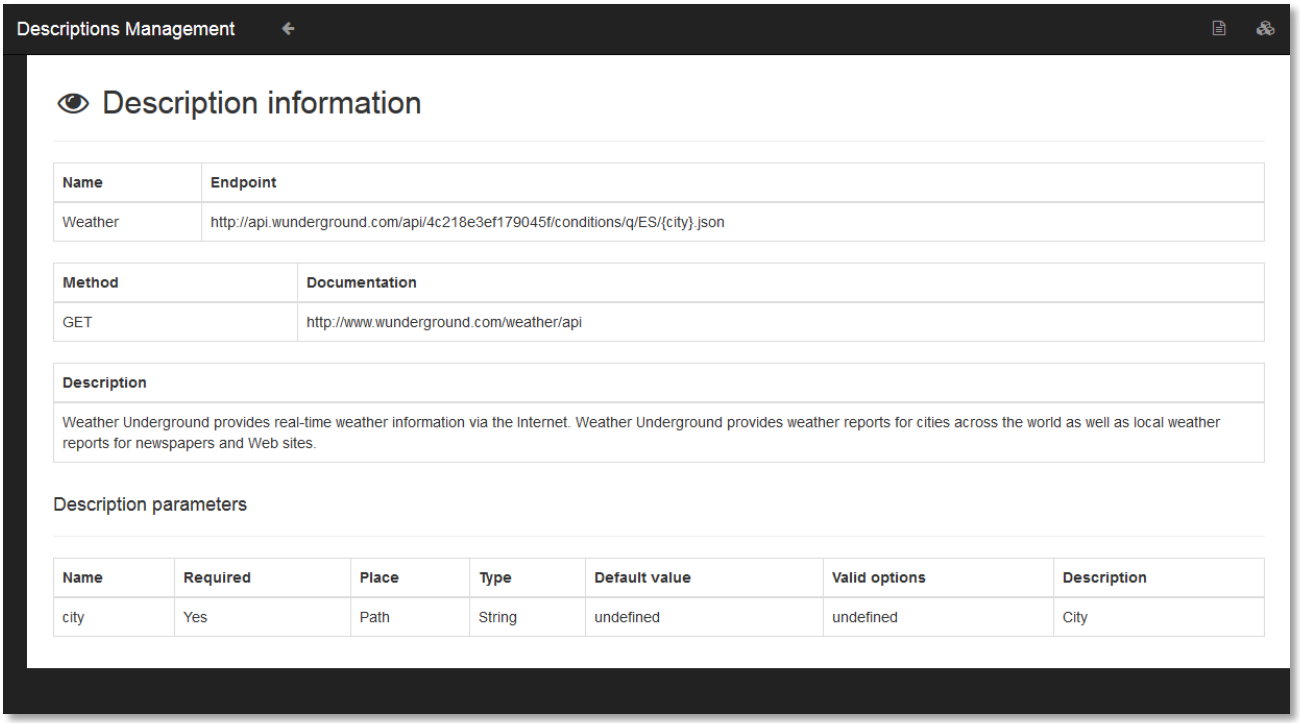


Figura 30: Visualización de una descripción

Manual de uso del módulo de composición de servicios

El módulo de composición de servicios se utiliza para crear servicios compuestos reutilizando otros servicios que tienen que haber sido descritos previamente en el módulo de gestión de servicios. Los servicios que se pueden utilizar aparecen en el lado izquierdo de la pantalla (ver figura 31) y pueden ser servicios simples u otros servicios compuestos. Las tareas que se pueden realizar son:

- añadir un servicio al área de composición
- conectar servicios
- usar el menú contextual
- eliminar un elemento del grafo
- eliminar un grafo
- etiquetar flechas
- ver información de un servicio
- configurar un servicio
- iterar un servicio
- ejecutar un servicio
- ejecutar la composición
- ver los resultados de ejecución de un servicio
- guardar la composición
- recuperar un servicio compuesto

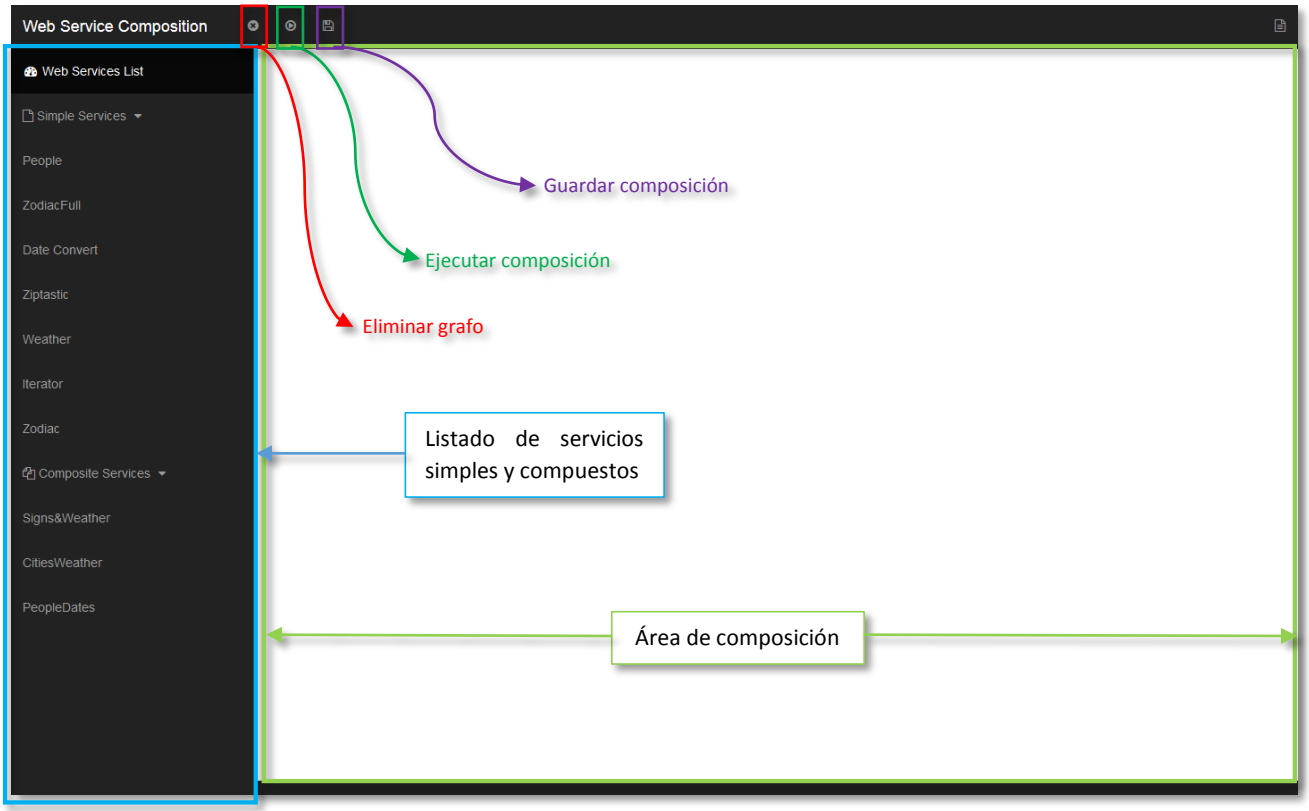


Figura 31: Pantalla principal del módulo de composición de servicios

A continuación se detalla como realizar cada una de estas tareas.

Añadir un servicio al área de composición

Para añadir un servicio al área de composición hay que arrastrar y soltar el nombre del servicio desde la lista de servicios hasta el área de composición. En cuanto se suelte el nombre aparecerá un nodo con el nombre del servicio añadido. El nodo que aparece representa al servicio y forma parte del grafo de composición.

Conectar servicios

Para conectar dos servicios se tiene que seleccionar en primer lugar el nodo origen y después seleccionar el nodo destino. En cuanto se seleccione el nodo destino aparecerá una flecha que conectará ambos nodos.

Menú contextual

Para usar el menú contextual hay que hacer click con el botón derecho del ratón sobre el elemento que deseemos. Tanto los nodos como las flechas tienen menú contextual. Este menú permite acceder a diferentes operaciones que se explicarán más adelante (ver figura 32).

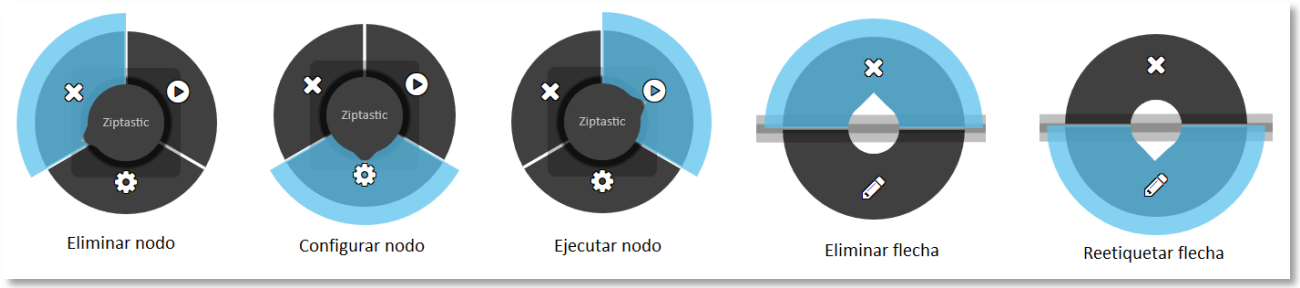


Figura 32: Menú contextual

Eliminar un elemento del grafo

Para eliminar un elemento del grafo, ya sea un nodo o una flecha, se debe abrir el menú contextual (ver figura 32) y seleccionar la opción de eliminar. En el caso de los nodos, aparecerá un cuadro de diálogo (ver figura 33) pidiendo confirmación. Si se pulsa sobre "Aceptar" el elemento se eliminará, y si se pulsa sobre "Cancelar" no se efectuará la eliminación.

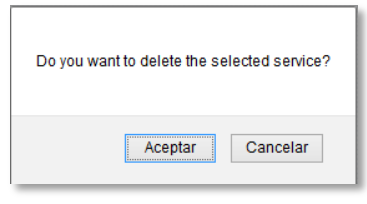


Figura 33: Diálogo de confirmación de eliminación de un servicio

Eliminar grafo

Para eliminar todo el grafo del área de composición solo hay que pulsar en el botón de borrar el grafo (ver figura 31). Al pulsar sobre el botón aparecerá un cuadro de diálogo (ver figura 34) pidiendo confirmación. Si se pulsa sobre "Aceptar" el grafo se eliminará, y si se pulsa sobre "Cancelar" no se efectuará la eliminación.

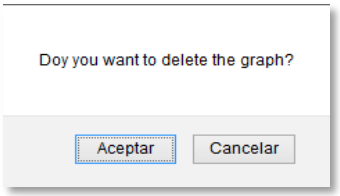


Figura 34: Diálogo de confirmación de eliminar grafo

Etiquetar flechas

Las flechas del grafo tienen una etiqueta por defecto que representa el identificador del resultado de la ejecución del nodo del que parten. Se puede cambiar esa etiqueta por otra más representativa abriendo el menú contextual de la flecha y seleccionando la opción de modificar la etiqueta (ver figura 32). Al pulsar sobre el botón aparecerá una ventana (ver figura 35) en la que se introducirá la nueva etiqueta. Si se pulsa sobre "Aceptar" la etiqueta se cambiará, y si se pulsa sobre "Cancelar" no se efectuará la modificación.

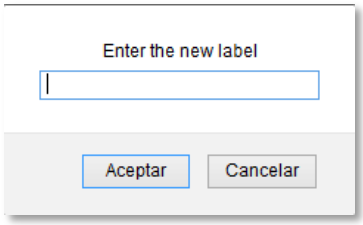


Figura 35: Ventana de modificación de etiqueta

Ver información de un servicio

Se puede ver la información de un servicio abriendo el menú contextual del servicio y seleccionando la opción de configuración (ver figura 32). Aparecerá una ventana dividida en tres pestañas: configuración del servicio, información del servicio y resultados. En la pestaña de información del servicio se pueden ver sus datos (ver figura 36).

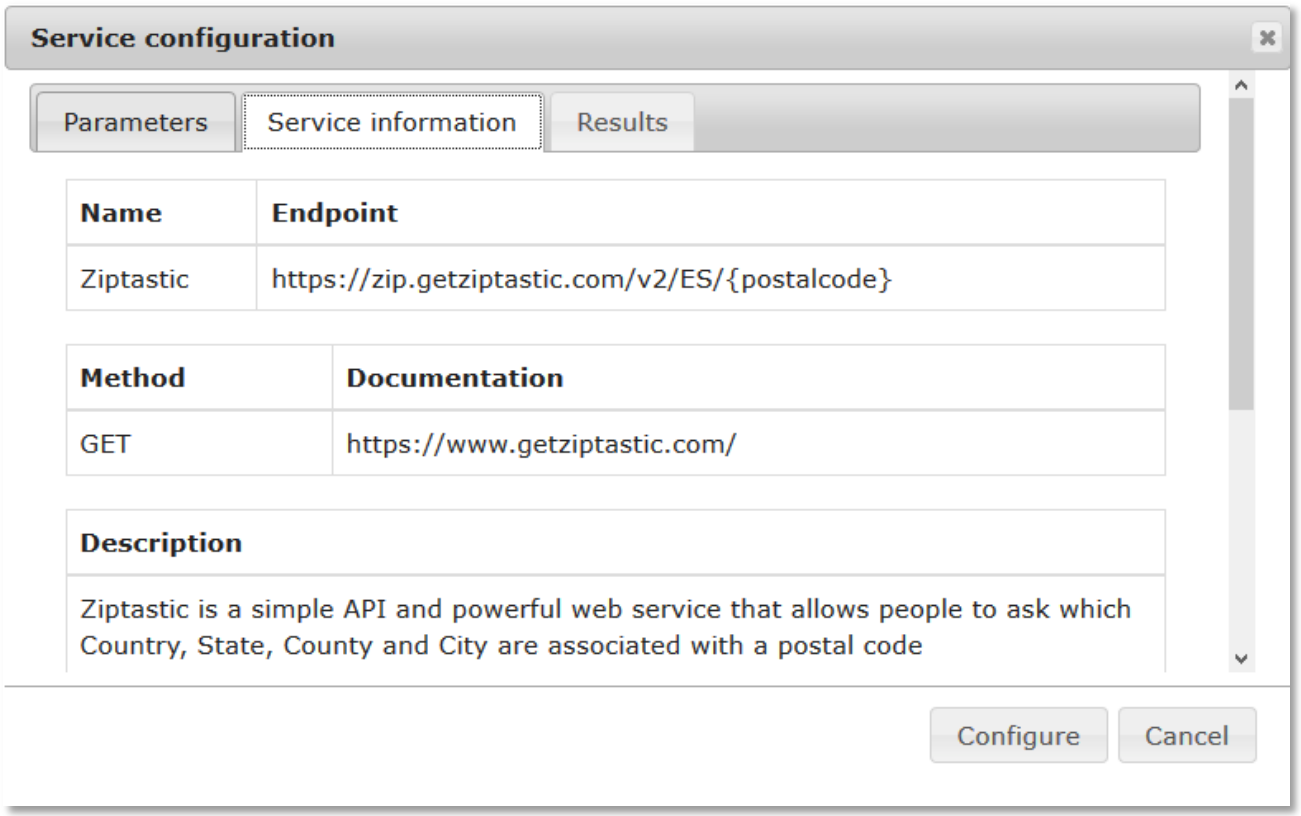


Figura 36: Pestaña de información de un servicio

Configurar servicio (sin iteración)

Para configurar un servicio se debe rellenar el formulario de configuración (ver figura 37) que aparece al abrir el menú contextual del servicio (ver figura 32). En el formulario aparecen todos los parámetros que tiene el servicio. Por cada parámetro se ha de indicar el origen del valor (manual o procedente de un nodo ejecutado) y el valor que se quiere que tome. Si se elige como origen un valor manual se ha de introducir directamente el valor en el campo correspondiente en el formulario. Si se elige como origen un nodo se ha de introducir una expresión en formato JSON para seleccionar el dato que debe tomar el parámetro. La expresión JSON deberá comenzar por el identificador de la flecha que hace referencia a los resultados de los que se quiere tomar el valor. Para que esto se vea más claro a continuación se muestra un ejemplo.

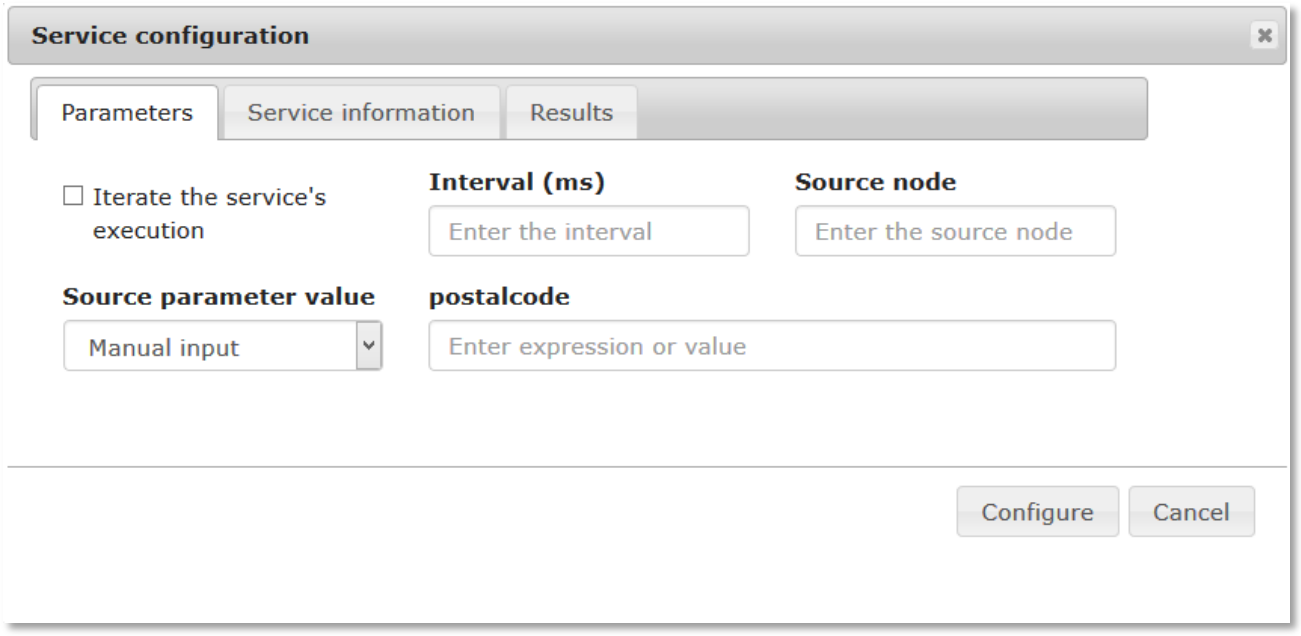


Figura 37: Pestaña de configuración de un servicio

A modo de ejemplo, en la figura 38 se ve un pequeño grafo de composición. El servicio Ziptastic ya está ejecutado y a partir de sus resultados (ver figura 39) se quiere configurar el servicio Weather (ver figura 40).

Figura 38: Pequeño grafo de ejemplo


```
{
   city: "Castellon De La Plana/Castello De La Pla",
   country: "ES",
   county: "Castellón",
   state: "Comunidad Valenciana",
   state_short: "VC",
   postal_code: "12006"
}
```


Figura 39: Resultado de ejecución del servicio Ziptastic

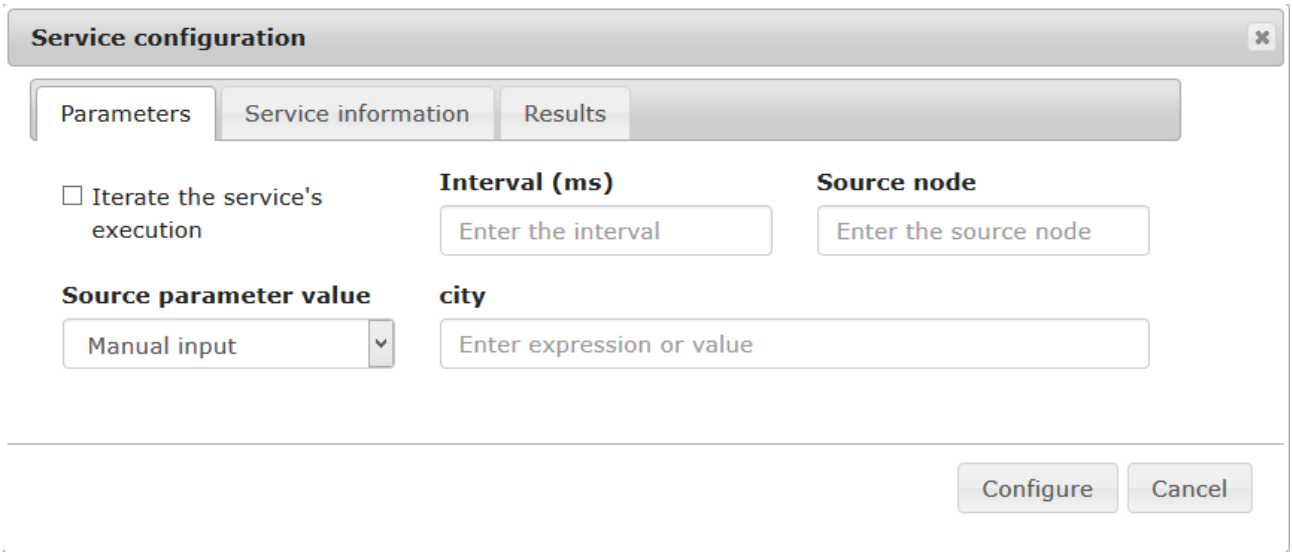


Figura 40: Pestaña de configuración del servicio Weather

Como se ve en la figura 40 el servicio Weather tiene un párametro "city" que representa el nombre de una ciudad. A partir de este parámetro el servicio obtiene el tiempo que hace en la ciudad introducida así como otros muchos valores (humedad, presión atmosférica, etc). Queremos obtener el valor del parámetro "city" a partir del resultado de ejecución del servicio Ziptastic que como se ve en la figura 28 entre otros valores tiene uno que se llama "city". En la figura 31 se ve que se ha etiquetado la flecha con el nombre de "cities" para darle un nombre representativo a los resultados del servicio Ziptastic que se quieren pasan al servicio Weather. Ahora para obtener el valor del parámetro "city" del servicio Weather se debe escribir una expresión JSON que empiece por el identificador del resultado del que se quiere obtener el valor, en este caso sería "cities" (la etiqueta de la flecha que representa los resultados del servicio Ziptastic) y a continuación lo que haga falta en formato JSON para seleccionar el dato correspondiente. En la figura 41 se ve el formulario de configuración del servicio Weather en el que se ve la expresión JSON que hace falta para seleccionar el valor "city" de los resultados del servicio Ziptastic.

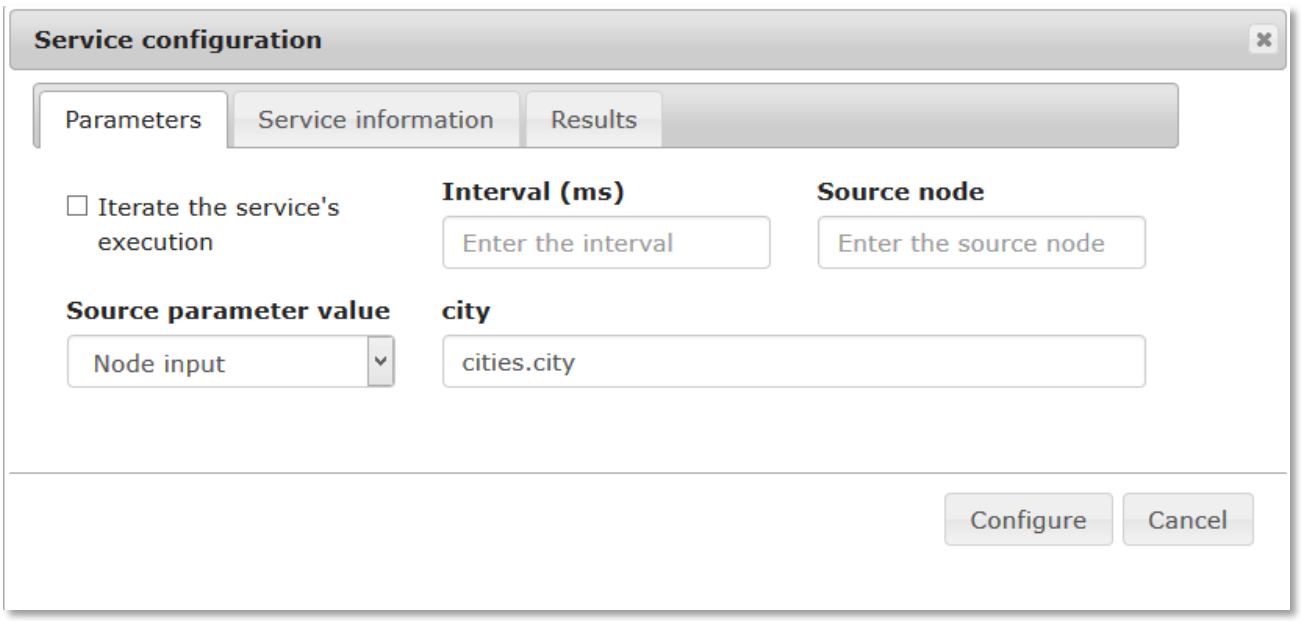


Figura 41: Configuración del servicio Weather

Una vez asignado el valor a los parámetros que se desee utilizar se pulsa en el botón "Configure" y la configuración del servicio se guarda. Cuando el nodo esté configurado aparecerá en color amarillo (ver figura 42).

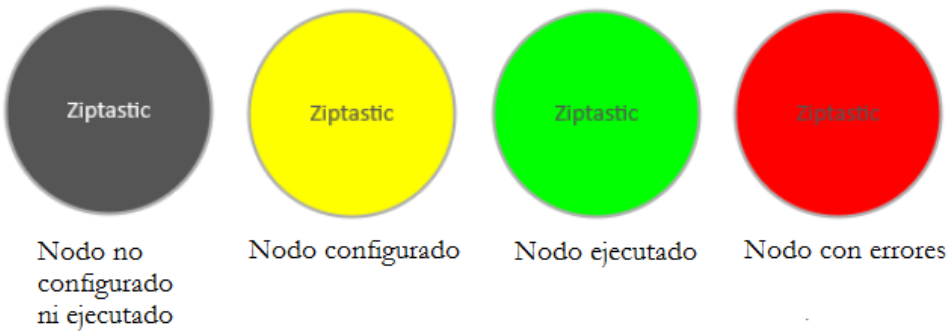


Figura 42: Posibles estados de los nodos del grafo

Configurar servicio (con iteración)

Para configurar un servicio para que itere a partir de los resultados de ejecución de otro nodo se deben seguir los mismos pasos que en la configuración sin iteración pero completando algunos campos más.

Cuando se quiere iterar un servicio significa que el resultado de ejecución del nodo del que obtienen los datos es una lista. Lo que se quiere es que el servicio que vamos a configurar se ejecute las veces que sean necesarias para obtener un resultado por cada dato recibido de la lista. Para hacer esto se debe marcar la casilla de verificación "Iterate the service's execution". Además se debe indicar la frecuencia de ejecución en milisegundos, es decir, el tiempo que se debe esperar entre llamada y llamada al servicio para que este no nos bloquee. Por último se debe rellenar el campo "Source node" que indica el nodo origen a partir de los resultados del cual se va a iterar.

Siguiendo con el ejemplo anterior, ahora los resultados del servicio Ziptastic son los que se muestran en la figura 43. Como se ve los resultados del servicio son una lista de JSONs. Lo que se pretende hacer con la iteración del servicio Weather es que calcule el tiempo de cada ciudad de la lista de resultados del servicio Ziptastic. Por tanto, deberemos rellenar los campos de iteración que se han explicado anteriormente y además escribir la expresión JSON como si se accediera a una lista. En la figura 44 se puede ver la configuración para que hacer que itere el servicio Weather y la expresión JSON correspondiente para obtener el valor de la ciudad.


```
[ {
    city: "Madrid",
    country: "ES",
    county: "Madrid",
    state: "Madrid",
    state_short: "MD",
    postal_code: "28014"
  },
  {
    city: "Barcelona",
    country: "ES",
    county: "Barcelona",
    state: "Cataluna",
    state_short: "CT",
    postal_code: "08013"
  },
  {
    city: "Valladolid",
    country: "ES",
    county: "Valladolid",
    state: "Castilla - Leon",
    state_short: "CL",
    postal_code: "47011"
  }]
```


Figura 43: Lista de resultados del servicio Ziptastic

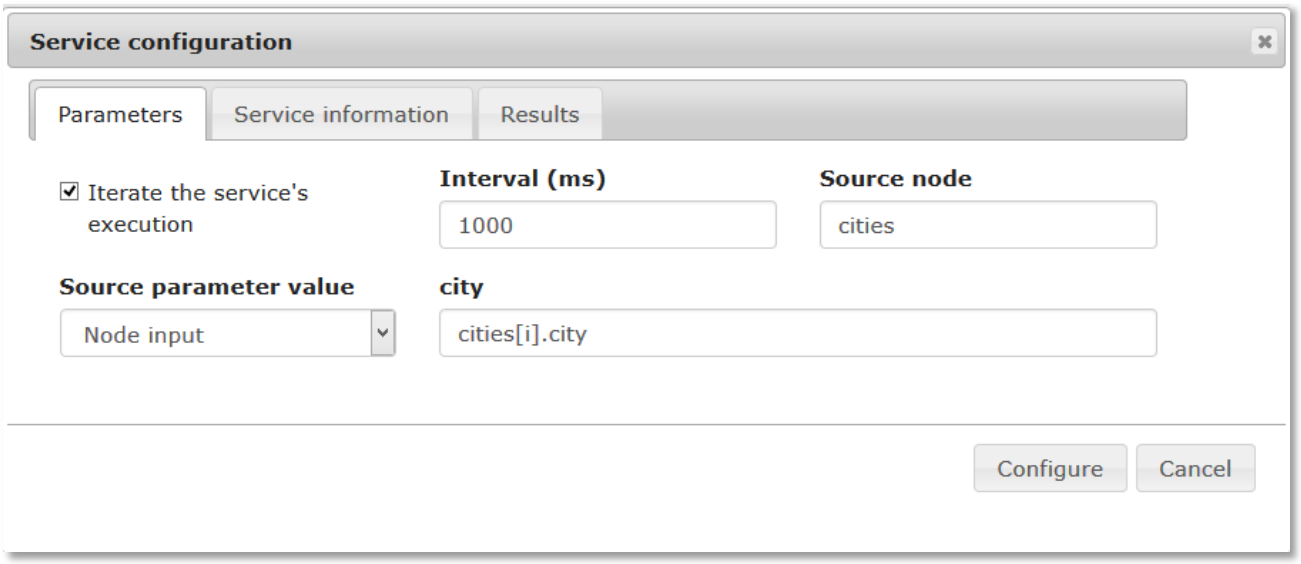


Figura 44: Configuración del servicio Weather para que itere

Ejecutar un servicio

Se puede ejecutar un servicio abriendo el menú contextual del servicio y seleccionando la opción de ejecución (ver figura 32). Si se produce algún error al intentar llamar al servicio se mostrará un mensaje avisando de ello y el servicio aparecerá en color rojo. Si la llamada al servicio es correcta el servicio aparecerá en color verde (ver figura 42).

Ejecutar la composición

Para ejecutar el servicio compuesto se debe pulsar en el botón de ejecutar composición (ver figura 31). Una vez pulsado el botón se irán ejecutando por orden todos los servicios que forman parte de la composición. En el caso de que haya algún problema al llamar a alguno de los servicios se mostrará un mensaje informando de ello y además tanto los servicios problemáticos como los que dependen de ellos aparecerán en color rojo.

Ver los resultados de ejecución de un servicio

Se puede el resultado de la ejecución de un servicio abriendo el menú contextual del servicio y seleccionando la opción de configuración (ver figura 32). Aparecerá una ventana dividida en tres pestañas: configuración del servicio, información del servicio y resultados. En la pestaña de resultados del servicio se pueden los resultados obtenidos (ver figura 46).

Guardar la composición

Para guardar un servicio compuesto se debe pulsar en el botón de guardar composición (ver figura 32). Aparecerá una ventana de diálogo (ver figura 45) en el que se deberá introducir el nombre que se quiere que tenga el nuevo servicio compuesto. Si todo va bien aparecerá un mensaje de confirmación y en caso contrario se informará del error.

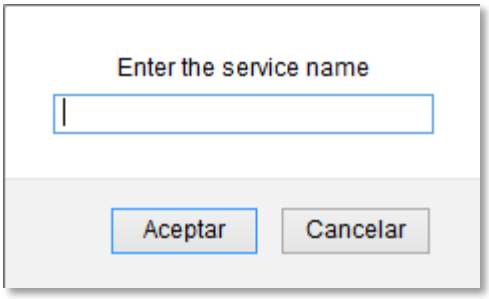


Figura 45: Diálogo para guardar un servicio

Figura 46: Pestaña de resultados del servicio Ziptastic

Recuperar un servicio compuesto

Los servicios compuestos guardados aparecen en la lista de servicios compuestos que se puede ver en la figura 31. Para recuperarlo simplemente se deberá seleccionar su nombre y arrastrarlo al área de composición tal y como se hace para incorporar un servicio simple. Después de hacer esto aparecerá en el área de composición el grafo que representa al servicio.